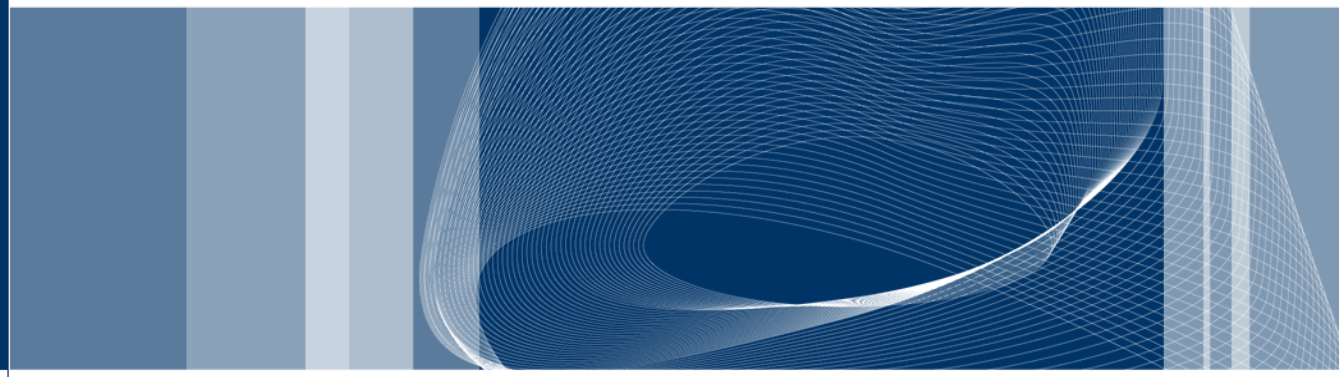
POLITECNICO DI MILANO

Natural Language Processing



Text Classification

Natural Language Processing

Some slide content based on textbook:
Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition *by Daniel Jurafsky and James H. Martin*

Contents

Preprocessing text for NLP applications

- Extracting plain text
- Tokenization
- Optional: cleaning and case-folding
- Optional: spelling correction
- Optional: Lemmatization/Stemming
- Optional: stop-word removal and low-frequency term removal

Building text classifiers

Building search indices

Preprocessing

No matter what NLP application we are working on

- if we are building a **search engine** where our documents come from a webcrawl,
- or if we are training a **phishing detector** on a collected set of email documents,

we will need to spend some time and effort investigating the source data and preprocessing it to convert it into a form that we can use for our application.

Extracting Plain Text

Extracting plain text

We may need to extract text from:

- textual documents: **.txt, HTML, e-mail**, etc.
  - usually discard mark-up (html tags) and other format-specific commands
  - in web crawl situations parser should be robust to badly formed HTML
- binary documents: **Word, PDF,** etc.
  - much more complex to handle,
  - for PDF documents, the structure of the text needs to be reconstructed.
  - If the PDF contains multiple columns, these may need to be recovered.
  - If all PDFs have same format, then rules for doing this could be hand-written, otherwise machine learning might be needed.
- images of **scanned documents**
  - requires specialized Optical Character Recognition (OCR) software that is now deep learning based
  - OCR is not perfect, so may introduce recognition errors in the text

Extracting plain text: Text Encondings

Note that there are various text encodings that could be used, which support different numbers of possible characters:

- ASCII -- traditional keyboard, only 128 characters
- UTF-8 -- handles 149k Unicode characters (https://en.wikipedia.org/wiki/Unicode) and works for 160+ languages,

Why do we need Unicode?

- to handle languages with:
  - non-latin character sets: Arabic, Cyrillic, Greek, Devanagari, etc.
  - special characters: like diacritical signs in Italian "Questa è così" and even in English: "Naïve"

Text Tokenization

Text Tokenization & Normalization

Many (if not all) NLP tasks require **tokenization**:

- **segmenting the text into sequences of characters called tokens**
- usually **tokens correspond to the words in the text**
- (although sometimes we tokenize at the character level)

So **tokenization** is **process of splitting up sentences into words**

- requires language-specific resources
- can be difficult for some languages (e.g. Chinese)

Preprocessing for information retrieval or text classification often also requires some form of **text normalization**:

1. normalizing word formats
2. segmenting sentences

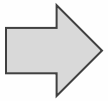
Space-based tokenization

Some languages use space characters between words: e.g. Arabic, Cyrillic, Greek, Latin, Devanagari, etc.,

- Can segment the text into tokens based on the white-space between words
- notethatitispossibletoreadasentenceinenglishwithoutspacesbetweenwords
  - so if we didn't have spaces we could work out some other way to tokenize text
  - But given hat they are available we may as well use them ;-)

- Problems with tokenizing certain texts
  - Dependiing on the application, it may make sense to split some hyphenated works like "Italian-style furniture"
  - Some languages are highly agglutinative, and can build very long and specific content
  - Some times the "unit of meaning" is spread over two non-hyphenated words

```
THE SONNETS by William Shakespeare
From fairest creatures we desire increase …
```



["THE", "SONNETS", "by", "William", "Shakespeare", "From", "fairest", "creatures", "we", "desire", "increase", …]

Issues in Tokenization

Can't blindly remove punctuation:

- m.p.h., Ph.D., AT&T, cap'n
- prices ($45.55)
- dates (01/02/06)
- URLs (http://www.stanford.edu)
- hashtags (#nlproc)
- email addresses (someone@cs.colorado.edu)

May need to deal with clitics: words that don't stand on their own

- "are" in we're, French "je" in j'ai, "le" in l'honneur

Should multiword expressions (MWE) be considered single words?

- New York, rock 'n' roll

Examples of simple Tokenizers

Default tokenizer in scikitlearn:

- uses the regular expression: token_pattern = '(?u)\b\w\w+\b'
  - where \b is matches word-boundaries (or start/end of string),
  - and \w = [a-zA-Z0-9] = any 'word' character

Tokenizer in NLTK (Natural Language Tool Kit in Python):

- uses a more complicated regular expression to catch various types of tokens:

```
pattern = r'''(?x)    # set flag to allow verbose regexps
    ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
  | \w+(-\w+)*        # words with optional internal hyphens
  | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
  | \.\.\.            # ellipsis
  | [][.,;"'?():-_`]  # these are separate tokens; includes ], [
'''
```

  - so the text: `text = 'That U.S.A. poster-print costs $12.40...'`
  - becomes: `['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']`

Tokenization in languages without spaces

Many languages such as Chinese don't use spaces to separate words!

- How do we decide where the token boundaries should be?
- Chinese characters are logograms, with each word contain 2.4 characters on average

Deciding what counts as word is difficult:

- Consider the sentence: 姚明进入总决赛 "Yao Ming reaches the finals"
- Could tokenize it at many levels:
  - 姚明 进入 总决赛
    YaoMing reaches finals
  - 姚 明 进入 总 决赛
    Yao Ming reaches overall finals
  - 姚 明 进 入 总 决 赛
    Yao Ming enter enter overall decision game
- So in Chinese it's common to just treat each character as a token.
- In other languages (like Thai & Japanese) complex segmentation is required.

Other options for text tokenization

Instead of **white-space segmentation** or **single-character segmentation**

- **use the data** to tell us how to tokenize
- with a **sub-word tokenization**
  - useful for splitting up longer words
  - and for allowing the machine learning model to learn explicity the morphology of the language
- we will use **Byte Pair encoding** to do this later in the course when we do deep learning.

Word Frequencies

How many words in a sentence?

- "I mainly do uh, mainly business data processing"

- "Seuss's cat in the hat is different from other cats!"
  - **Lemma**: same stem, part of speech, rough word sense
    - cat and cats = same lemma
  - **Wordform**: the full inflected surface form
    - cat and cats = different wordforms

She actively encouraged the discouraged child to act courageously and confront her fears.

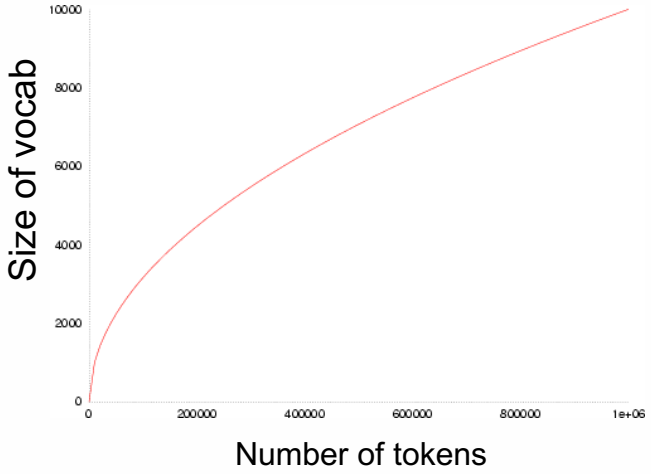
- **Type**: an element of the vocabulary: she, actively, encouraged, …
- **Token**: an instance of that type in running text.

Statistical Laws of Text

Heap's law: https://en.wikipedia.org/wiki/Heaps'_law

- Vocabulary grows with approximately the square root of document / collection length:

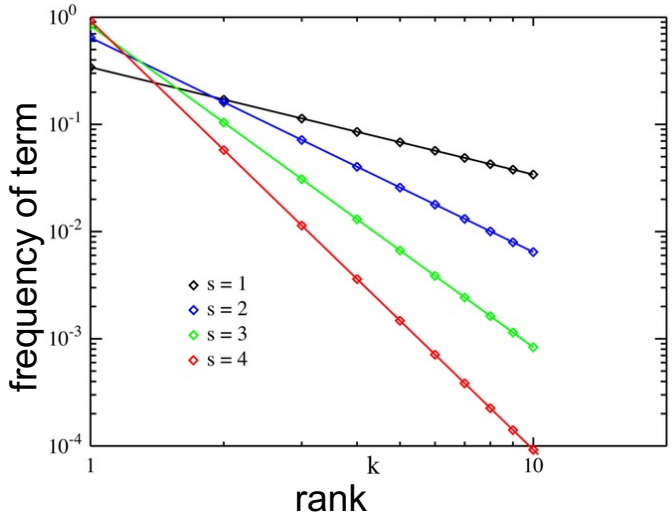
$$V(l) \propto l^{\beta}$$
$$\beta \approx .5$$



Zipf's law: https://en.wikipedia.org/wiki/Zipf%27s_law

- A term's frequency in a collection is approximately proportional to the inverse of its rank

$$\mathrm{ctf}_t \propto \frac{1}{\mathrm{rank}(t)^s}$$
$$s \approx 1$$



Monkeys with Typewriters

	Tokens	Vocabulary
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884 thousand	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

Heap's law derives from Zipf's law:

- it can be explained by a **random typing model** with random space placement,
- a.k.a. Monkeys at typewriters
  - https://www.jstor.org/stable/1421275
  - https://www.cs.cmu.edu/~zollmann/publications/monkeypaper.pdf

And if you wait long enough with a sufficient number of monkeys, eventually one of them will produce Shakepeare

- Infinite monkey theorem
  - https://en.wikipedia.org/wiki/Infinite_monkey_theorem

Text Normalisation

Word Normalization

- Process of putting words/tokens in a standard format
  - U.S.A. or USA?
  - uhhuh or uh-huh?
  - Fed or fed?
  - am, is, be, are?
- Critical for Web Search applications, otherwise a query for ”USA” might not return documents containing the term “U.S.A”

Case folding

Who doesn't love ABBA?
⇒ who doesn't love abba?

Applications like **web search** often reduce all letters to lower case

- drastically reduces size of vocabulary and increases recall (set of valid documents found)
- users tend to use lowercase in search queries anyway

For classification problems, removing case

- reduces vocabulary and thus number of parameters that must be learnt
- helping classifier to generalise well from far fewer examples

Problem: often lose important information by removing cases.

- For example, in text the word "who" might refer to:
  - the WHO (the World Health Organization), the Who (the rock band) or a person who was just being talked about
- Another example is "us" which might have been:
  - the US or a first person plural
- Thus retaining case can be helpful for many applications:
  - like sentiment analysis, Machine Translation, Information extraction.

Morphology

Morphology

Fancy word from linguistics that refers to:

- analysis of **structure** of **words**

Morpheme: smallest linguistic unit that has semantic meaning

- e.g.: unbelievably → un-believe-able-ly
- morphemes are divided into:
  - **root**: the base form (*believe*)
  - affixes: **prefix** (*un-*), **infix** (*-able-*), or **suffix** (*-ly*)

Lexicon

Morphemes compose to make *lexemes*

- **Lexeme**: unit of lexical meaning that exists regardless of the number of inflectional endings it may have or the number of words it may contain
  - E.g.: BELIEVE, NEW YORK, RUN
- **Lemma**: canonical form of a lexeme
  - E.g.: TO RUN
- **Lexicon**: set of lexemes
  - Lexicons for NLP usually contain affixes and other info
- A **word** is, in general, an inflected form of a lexeme
  - E.g.: *unbelievably*, *runs*

Composing morphemes

- Morphologic rules:
  - Restrict the ordering of morphemes (morphotactics)
  - E.g.: PLURAL NOUN = SINGULAR NOUN + PL
- Orthographic rules:
  - aka "spelling rules" or "two-level rules"
  - E.g.: fox + s → foxes; un-believe-able-ly → unbelievably
- Lexicons in NLP (when used for morphology):
  - Define base forms
    - E.g.: fox: NOUN, SINGULAR, …
  - Address irregular forms
    - E.g.: wrote → root: write; mice → root: mouse
  - Define affix morphemes
    - E.g.: PL → s

Lexicon syntax

Lexicon syntax is little cryptic/complicated.
Here are some examples:

```
\lf `mouse
\lx N
\alt Suffix
\fea irreg
\gl1

\lf `mice
\lx N
\alt Clitic
\fea pl irreg
\gl1 `mouse
```

Has irregular plural

Is the irregular plural form of "mouse"

```
\lf `fox
\lx N
\alt Suffix
\gl1
```

Part-of-speech: Noun

```
\lf +s
\lx INFL
\alt End
\fea v/v s
\gl1 +3SG
\gl2 +3SG

\lf +s
\lx IC_SUFF
\alt Clitic
\fea n-aj/n reg
\gl1 +PL
\gl2 +PL
```

+s: 3rd singular person

+s: plural

Recognition: with morphologic rules

- **`Foxes`** → **`fox+PL`**, **`noun`**
  - **`fox+PL`** disambiguated, as **`fox`** is a noun, and nouns have +PL but not +3SG
- **`Talk`**
  - Can be noun or verb
  - No disambiguation!
  - Context is needed
- POS taggers are needed
  - These tools leverage the context

FreeLing

- Multilanguage (Spanish, English, Italian)
- Based on a statistical model
  - It is actually a POS tagger (more on that later…)
  - Can analyze the morphologic structure of the word (affixes, …), for some languages (try "*prendimelo*")
  - Can return the base-form and POS
- Modern approach: we have lots of space to store all inflexed forms…

  - FreeLing: http://garraf.epsevg.upc.es/freeling

    http://nlp.lsi.upc.edu/freeling/demo/demo.php

Lemmatization

Represent all words as their lemma, their shared root
= dictionary headword form:

- *am, are, is* → *be*
- *car, cars, car*'s, *cars*' → *car*
- Spanish quiero ('I want'), quieres ('you want')

→ querer 'want'

- *He is reading detective stories*

→ *He be read detective story*

Lemmatization is done by Morphological Parsing

- Morphemes:
  - The small meaningful units that make up words
  - Stems: The core meaning-bearing units
  - Affixes: Parts that adhere to stems, often with grammatical functions
- Morphological Parsers:
  - Parse *cats* into two morphemes *cat* and *s*
  - Parse Spanish *amaren* ('if in the future they would love') into morpheme *amar* 'to love', and the morphological features *3PL* and *future subjunctive*.

Lexicon-free methods: stemming

Simple algorithms: no lexicon needed!

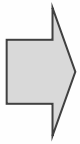
- often used in Information Retrieval to reduce computational requirements

- Why is it useful for Information Retrieval?
  - Ideally: assign unique ID (a *stem*) to all inflected forms of a given base form
    - E.g.: (dog, dogs, doggy) → ID1, (cat, cats, catlike) → ID2, …
  - Apply to the document collection:
    - Doc1: "… dog…" → doc1: "… ID1 …"
      Doc2: "… doggy…" → doc2: "… ID1 …"
  - And to the query:
    - "dogs" → "ID1"; the system finds [doc1, doc2]

Stemming

- Reduce terms to stems, chopping off affixes crudely

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.



Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note.

Porter Stemmer

- Based on a series of rewrite rules run in series
  - A cascade, in which output of each pass fed to next pass

- Some sample rules:

$$\begin{aligned}
\text{ATIONAL} &\rightarrow \text{ATE} \quad \text{(e.g., relational} \rightarrow \text{relate)} \\
\text{ING} &\rightarrow \epsilon \quad \text{if stem contains vowel (e.g., motoring} \rightarrow \text{motor)} \\
\text{SSES} &\rightarrow \text{SS} \quad \text{(e.g., grasses} \rightarrow \text{grass)}
\end{aligned}$$

Lexicon-free methods: Porter

Porter Stemming Algorithm ('80)

- A set of *rewriting rules*
  - E.g.: +ATIONAL → +ATE (es: relational → relate),
    +ING → ε (es: motoring → motor), …
- Simple, but error prone
  - Collisions (Two different words, the same stem)
    Word: Policy → stem: police
    Word: Police → stem: police
    Searching for "policy" I can also find docs containing "police"

- Porter Stemmer for the English language http://www.tartarus.org/~martin/PorterStemmer/
- Snowball (programming language for stemmers) http://snowball.tartarus.org/

Dealing with complex morphology is necessary for many languages

- e.g., the Turkish word:
- Uygarlastiramadiklarimizdanmissinizcasina
- `(behaving) as if you are among those whom we could not civilize'
- Uygar `civilized' + las `become'

  + tir `cause' + ama `not able'

  + dik `past' + lar 'plural'

  + imiz 'p1pl' + dan 'abl'

  + mis 'past' + siniz '2pl' + casina 'as if'

Stemming vs lemmatization

the boy's cars are different colors ⇒
the boy car be differ color

In **information retrieval** to prevent vocabulary mismatch between query and document:

- perform stemming (or lemmatization) before adding terms to the index

  car, cars, car's, cars' ⇒ car

- Difference between Stemming and Lemmatization?
  - Stemming = Simple algorithm that applies rules to extract word stems.
    - See for example Porter's stemmer
    - http://snowball.tartarus.org/algorithms/porter/stemmer.html
  - Lemmatization = More sophisticated NLP technique
    - uses vocabulary and morphological analysis to extract the lemma

- Very important for morphologically rich languages (e.g. French)
- Trade off:
  - Stemming increases recall, but loses information and can lower precision
  - If possible best to include both pre and post-stemmed terms in search index.

Sentence Segmentation

The punctuation marks: "!" and "?" are mostly unambiguous and indicate the end of the statement/question

- except for maths expressions like: "5! = 120"
- or statements containing unknowns: "Fill in the missing values: 1, 2, ?, 4, 5, ?, 7, 8, ?,10

Period "." is commonly used to end a sentence, but is very ambiguous, since it also appears in

- abbreviations like Inc. or Dr.
- and numbers like .02% or 4.3

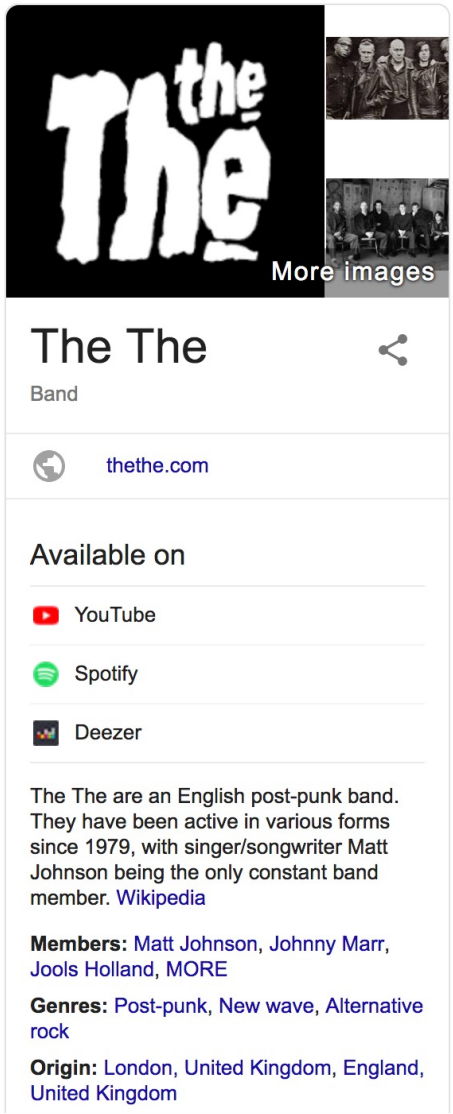
Common algorithm: Tokenize and then use rules or ML to classify a period as either (a) part of the word or (b) a sentence-boundary.

Stopwords

Word
the
be
to
of
and
a
in
that
have
I
it
for
not
on
with
he
as
you
do

Stopword lists

- Stopwords are simply the most frequent terms in language.
  - extremely high document frequency scores (low discriminative power)
  - convey very little information about the topic of the text
- Removing stopwords can sometimes boost performance of retrieval/classification models
  - More likely it will just reduce computational / memory burden
  - plus speeds up index by removing massively long posting lists
- Problem: sometimes stopwords are useful!
  - consider the Rockband "the The"
  - or "a white house" vs "the white house"



Spelling Correction

Error correction

There can be many sources of errors in an extracted text file:

- Errors introduced by Optical character recognition (OCR)
  - especially for handwriting recognition
  - in the form of substitutions, deletions, addition of spaces, etc.
- Human errors:
  - Spelling mistakes (spelling errors)
  - Typos (linked to the keyboard, replacement, insertion, deletion, transposition)
  - Cognitive errors (phonetic errors due to homonyms)
- Deliberate use by the author of non-existent (made-up) words.

Depending on the application, we might need to discover and correct theses errors as part of the pre-processing pipeline…

Probabilistic Spelling Correction

Consider the sentence:

... stellar and versatile acress *whose combination ...*

- observed word "***acress***" is **not** a **valid word** in our **vocabulary**

What is the chance that writer intended to write "***actress***" instead?

$P(correct = \text{“actress”} \mid observed = \text{“acress”})$

- If we had an ENORMOUS corpus of text
  - with every possible misspelled word
  - where authors had corrected every mis-spelling they'd ever written
  - we could just **estimate the relative frequency**:

$$P(correct=\text{“actress”} \mid observed=\text{“acress”}) = \frac{\#(correct=\text{“actress”} \ \&\ observed=\text{“acress”})}{\#(observed=\text{“acress”})}$$

- But we don't have such a corpus ☹
  - so need to work out some other way to estimate the probability...

Spelling Correction: edit distance

We could estimate the probability using the string **edit distance**.

- count number of insertions, deletions, substitutions or transpositions needed to get from one string to the other

Error	Correction	Correct Letter	Error Letter	Position (Letter #)	Type
acress	actress	t	–	2	deletion
acress	cress	–	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	–	2	5	insertion
acress	acres	–	2	4	insertion

- then **actress, caress, cress, access, across, acres** would all be equally likely
- but surely some of those words (like **actress**) are more likely than others (like **cress**)?

Spelling Correction: Bayes Rule

We can use Bayes' Rule to write the condition the other way around:

$$P(\text{correct}=\text{“actress”} \mid \text{observed}=\text{“acress”}) = \frac{P(\text{correct}=\text{“actress”}, \text{observed}=\text{“acress”})}{P(\text{observed}=\text{“acress”})}$$

$$= \frac{P(\text{observed}=\text{“acress”}|\text{correct}=\text{“actress”})\, P(\text{correct}=\text{“actress”})}{P(\text{observed}=\text{“acress”})}$$

Note that the denominator is the same for all candidate corrections

- so we can ignore it and normalise probabilities later

$$P(\text{correct}|\ \text{observed}) \propto P(\text{observed}=\text{“acress”}|\text{correct}=\text{“actress”})\, P(\text{correct}=\text{“actress”})$$

Likelihood of correction
How likely is it that the author acidentaly typed the observed word when they intended to type the corrected one?

Prior probability of corrected word
How frequent is that word overall in English?

Spelling Correction: parameter estimation

For each possible correction, need to estimate:

- the likelihood, *P(observed|correct)*
- and the prior, *P(correct)*

For the prior

- see how popular that word is in a large corpus
- In corpus of *N=44* milions of words we have:

c	*freq(c)*	*P(c)*
actress	*1343*	*.0000315*
cress	*0*	*.000000014*
caress	*4*	*.0000001*
access	*2280*	*.000058*
across	*8436*	*.00019*
acres	*2879*	*.000065*

$$P(c) = \frac{C(c) + 0.5}{N + 0.5}$$

Smoothed estimate of prior probability

Spelling Correction: parameter estimation

For the likelihood:

- estimate $P(observed="acress"|correct="across")$ by counting in large corpus of errors how many times e has been observed instead of o
- confusion matrix (del, ins, sub, trans) – **need corpus annotated with corrections**

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X	Y (correct)																								
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

Extensions: less labelled data

- Previous approach for estimating $P(observed \mid correct)$:
  - required large annotated corpus to create confusion matrices
- Another approach is to compute matrices by iteratively using spelling error correction algorithm (Kernighan et al., 1990)
  1. initializes matrices with equal values
  2. run spelling error correction algorithm on set of spelling errors
  3. set of typos is paired with their corrections, confusion matrices can be recomputed
  4. goto 2, unless some termination condition is true
- Is an instance of the EM algorithm (Dempster et al.,1977)

Context is needed

c	freq(c)	p(c)	p(t|c)	p(t|c)p(c)	%
actress	1343	.0000315	.000117	3.69×10^{-9}	**37%**
cress	0	.000000014	.00000144	2.02×10^{-14}	**0%**
caress	4	.0000001	.00000164	1.64×10^{-13}	**0%**
access	2280	.000058	.000000209	1.21×10^{-11}	**0%**
across	8436	.00019	.0000093	1.77×10^{-9}	**18%**
acres	2879	.000065	.0000321	2.09×10^{-9}	**21%**
acres	2879	.000065	.0000342	2.22×10^{-9}	**23%**

44%

- Bayesian algorithm predicts:
  - acres (44%) as first choice
  - and actress (37%) as second choice
- Clear from **context** that the correct word was **actress** not **acres**:
  ... was called a "stellar and versatile acress whose combination of sass and glamor has defined her ..."

Making use of the context

Clear from **context** that the correct word was **actress** not **acres**:

> ... *was called a "stellar and versatile* acress *whose combination of sass and glamor has defined her ..."*

How can we make use of that context information?

- look at **the preceding words**
  - and see how much they agree with the candidate correction
  - here "**versatile actress**" sounds much more likely than "**versatile acres**"
- we can count in a large corpus of documents the frequency of the bigrams
  - and replace the **unigram** probability P(correct="actress") with the **bigram** probability P(bigram="versatile actress")
  - at which point the spelling correction is making use of both the **observed word** and the **observed context**

Leads to a Naïve Bayes classifier

By replacing the unigram probability P(correct="actress") with the bigram probability P(bigram="versatile actress")

- We are effectively estimating a Naïve Bayes model
- Consider that:

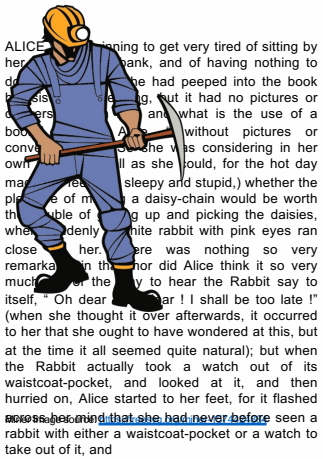
$$P(\text{bigram}=\text{"versatile actress"}) = P(\text{previous}=\text{"versatile"}, \text{correct}=\text{"actress"}) = P(\text{previous}=\text{"versatile"} \mid \text{correct}=\text{"actress"})\, P(\text{correct}=\text{"actress"})$$

- So in the end we have:

$$P(\text{correct} \mid \text{observed}, \text{previous}) \propto P(\text{observed}=\text{"acress"} \mid \text{correct}=\text{"actress"})\, P(\text{previous}=\text{"versatile"} \mid \text{correct}=\text{"actress"})\, P(\text{correct}=\text{"actress"})$$

- Which is a form of Naïve Bayes model with two features, the observed (incorrect) word and the previous word in the sentence….

Building Text Classifiers



Building Text Classifiers

- Revision: What is Machine Learning?
- What is text classification?
- Why would I want to classify text?
- Pre-processing text
- Training a model
- Different types of models
- Evaluating a text classifier
- Conclusions

Revision: What is Machine Learning?

Machine Learning (ML)

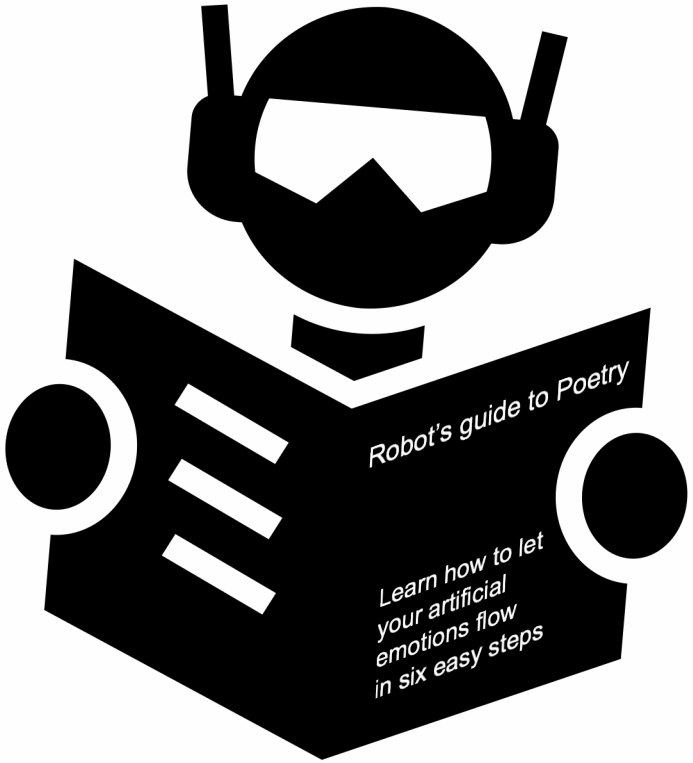
Techniques aimed to make machines "act more intelligent" by generalising from past data to predict future data

Standard definition for ML:

> "A computer program is said to learn from ***experience E*** with respect to some class of ***tasks T*** and a ***performance measure P***, if its performance at tasks in *T*, as measured by *P*, improves because of experience *E*."

- Tom M. Mitchell
https://en.wikipedia.org/wiki/Tom_M._Mitchell

image source: https://www.ml.cmu.edu/people/core-faculty.html

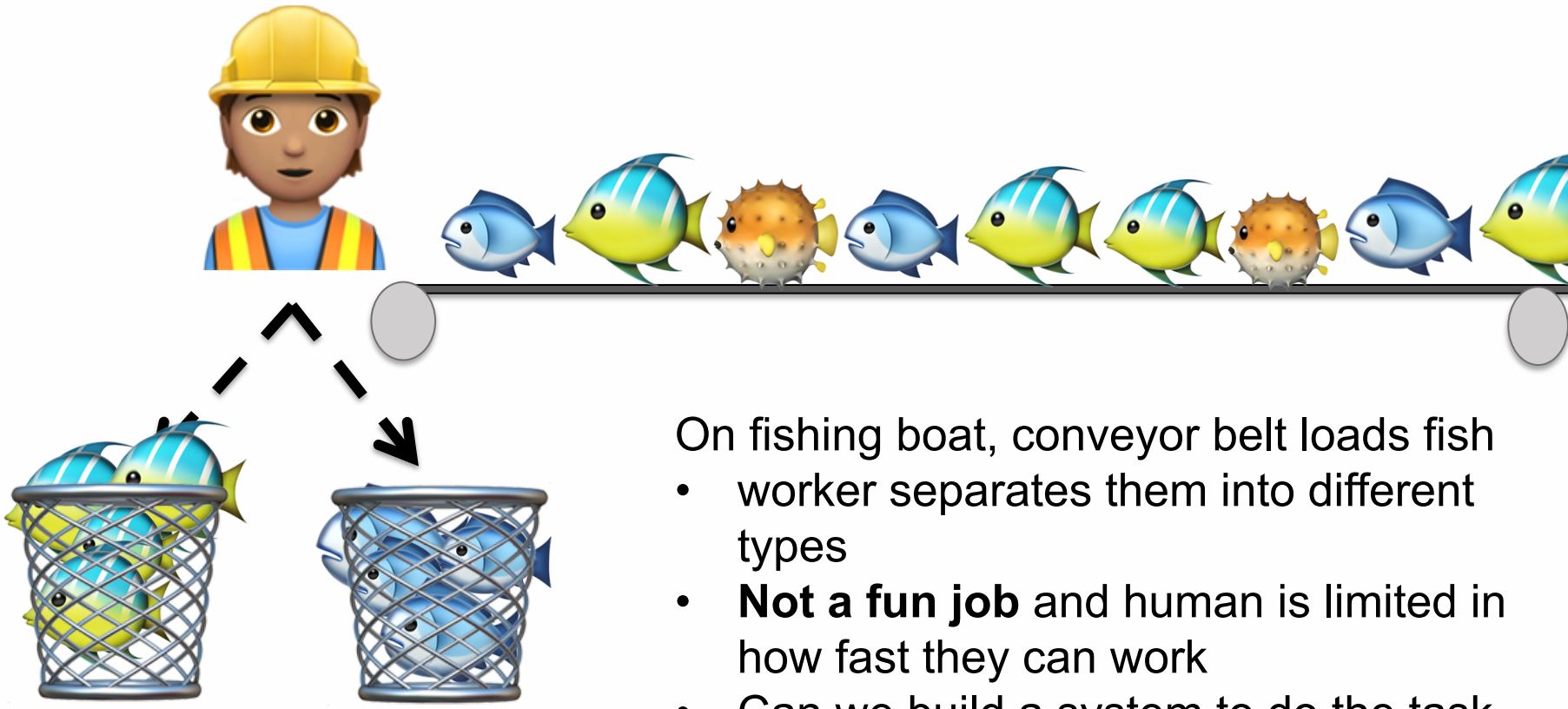


CC reading robot by Mavadee, TH, source:
https://thenounproject.com/search/?q=robot%20reading&i=2401182

Think that a robot learning to express its emotions is silly? Then read this article:
"A robot wrote this entire article. Are you scared yet, human?"
https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3

Supervised Machine Learning

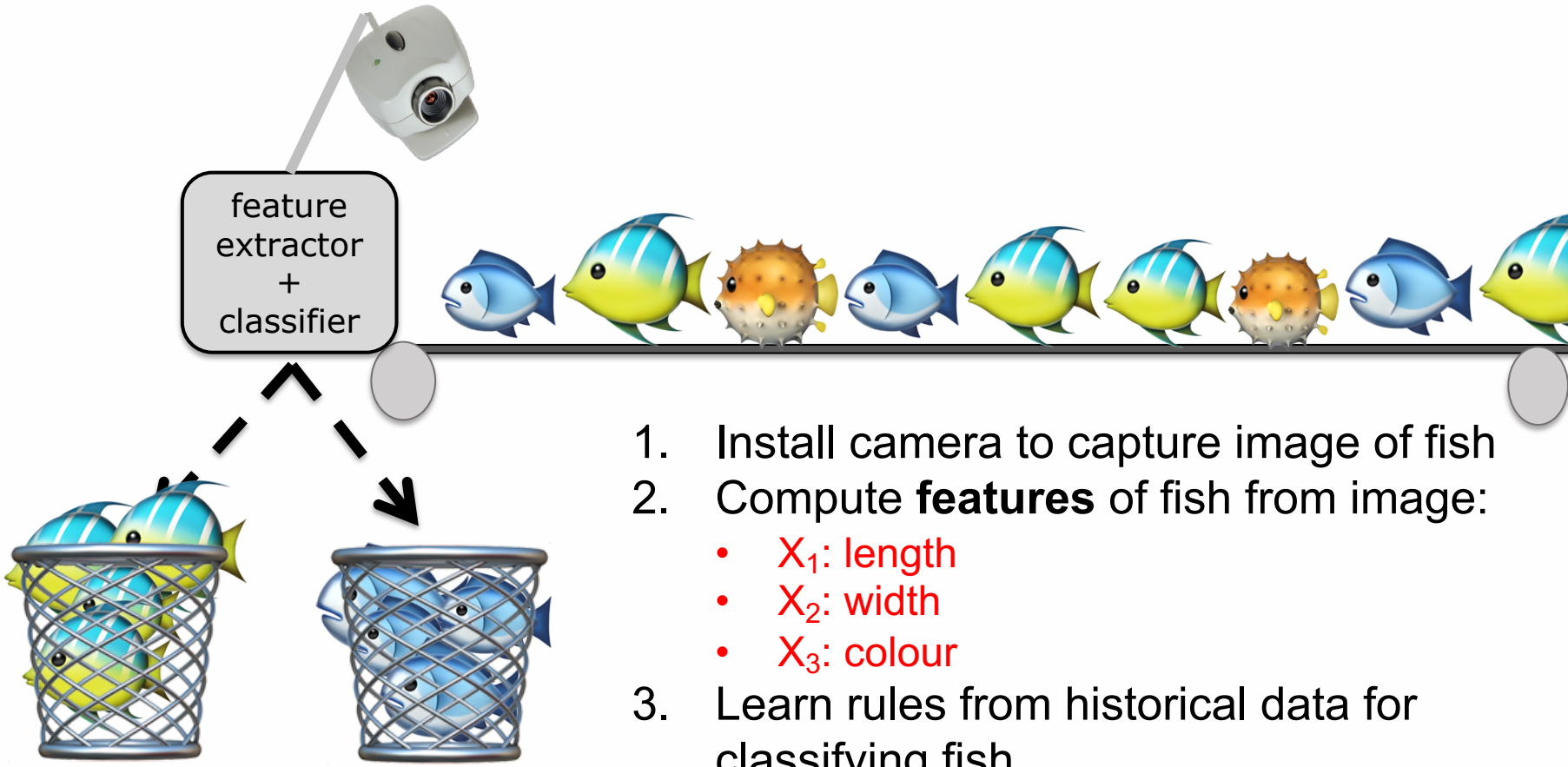
Motivating example: classification task



On fishing boat, conveyor belt loads fish

- worker separates them into different types
- **Not a fun job** and human is limited in how fast they can work
- Can we build a system to do the task automatically?

Motivating example: install sensors



1. Install camera to capture image of fish
2. Compute **features** of fish from image:
   - X_1: length
   - X_2: width
   - X_3: colour
3. Learn rules from historical data for classifying fish

   if (X_1/X_2<1.4 and X_3=blue) => **bass**

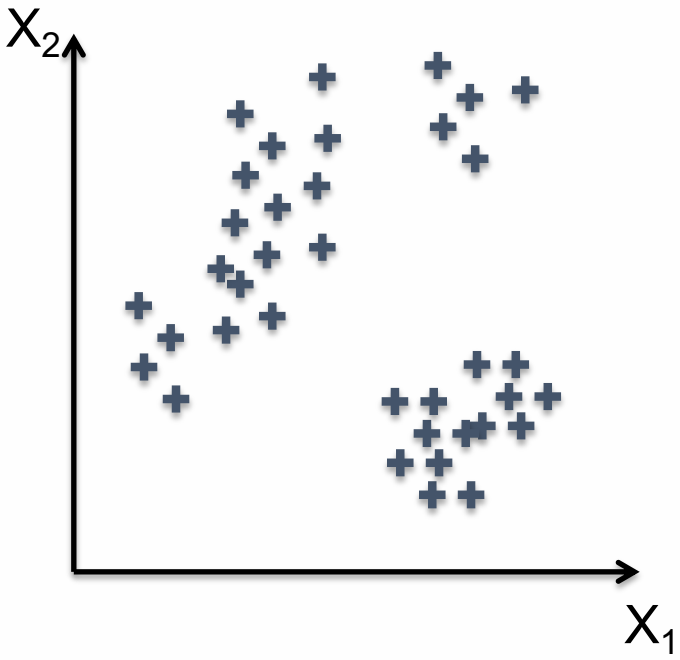
   else if (X_1/X_2>2.1 and X_3=pink) => **tuna**

   else => **unknown**

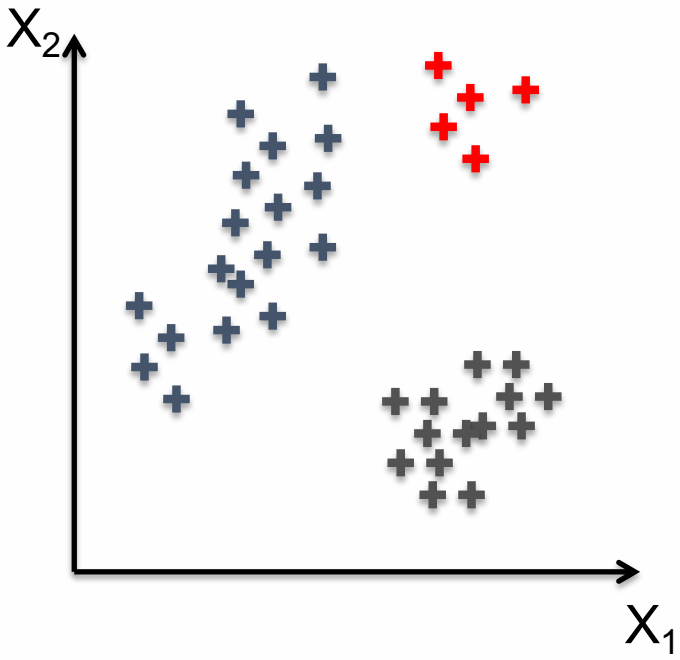
Unsupervised vs Supervised Learning

- Each training instance (fish) is a point in some feature space

UNSUPERRVISED



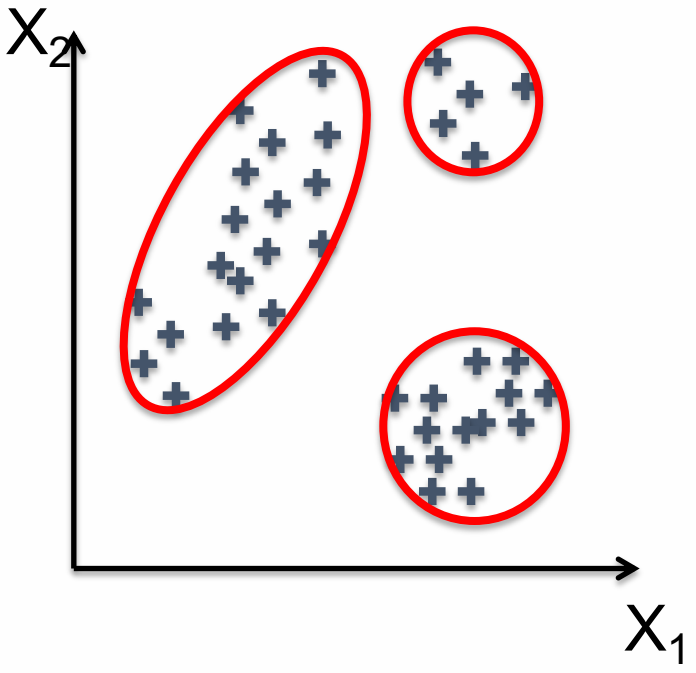
SUPERVISED



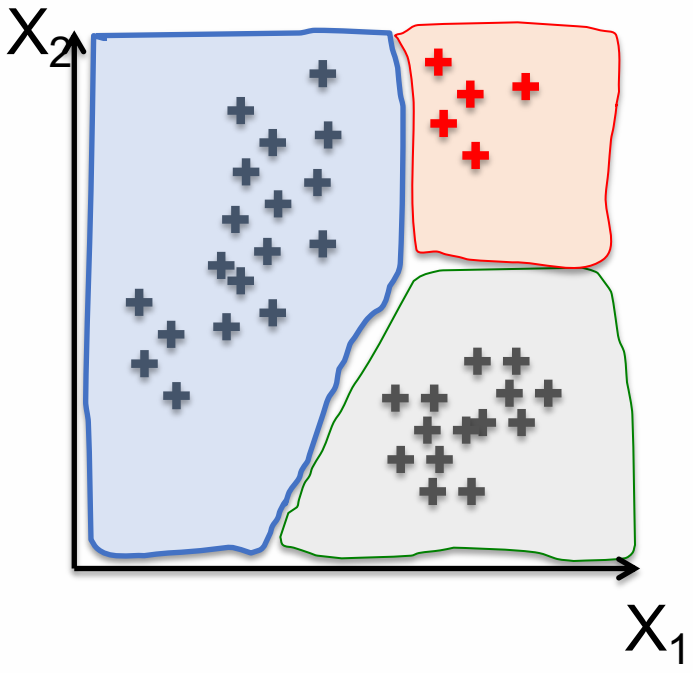
- In an supervised learning problem each training data point has been labeled (we know the type of fish)

Unservised vs Supervised Learning

UNSUPERRVISED

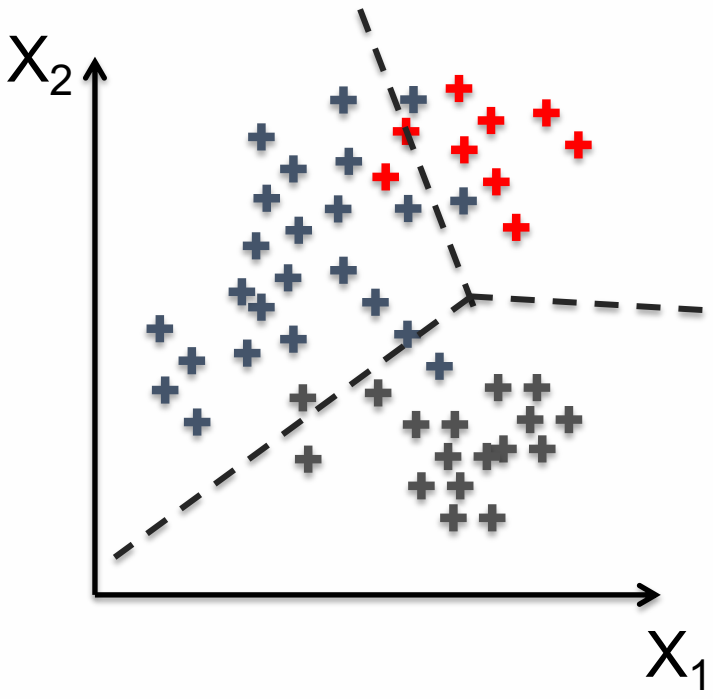


SUPERVISED



- Unsupervised problem: label the instances
- Supervised problem: partition the space for future predictions

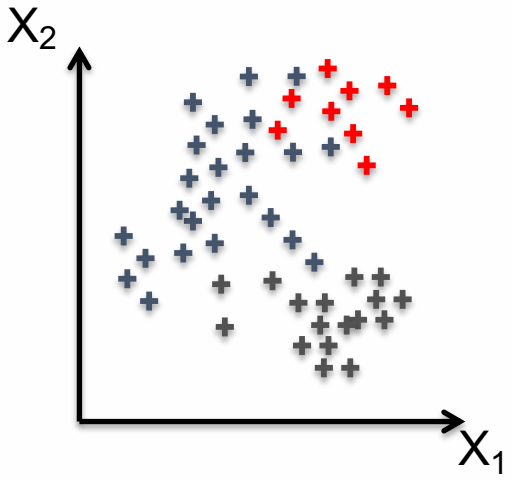
In practice



Data is usually overlapping

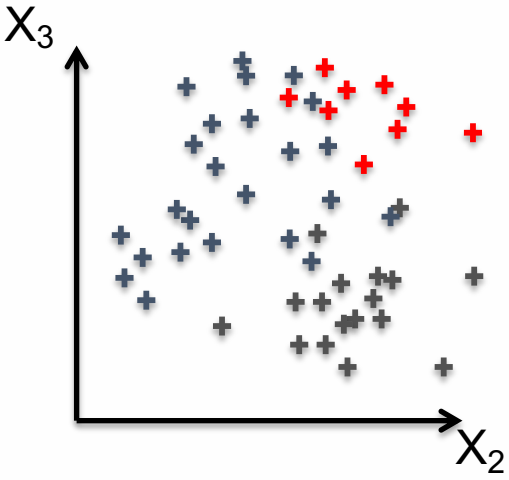
- so classes are not linearly separable

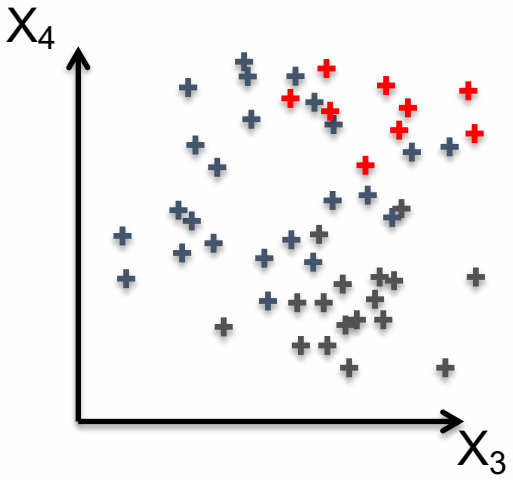
In practice



Instances we have many more than two dimensions

- some dimensions may be better at distinguishing the classes than others

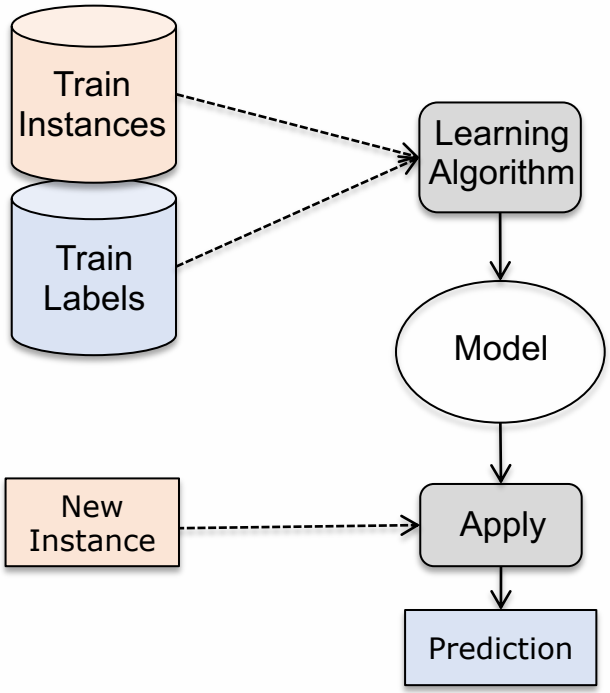




Example of a Data Set

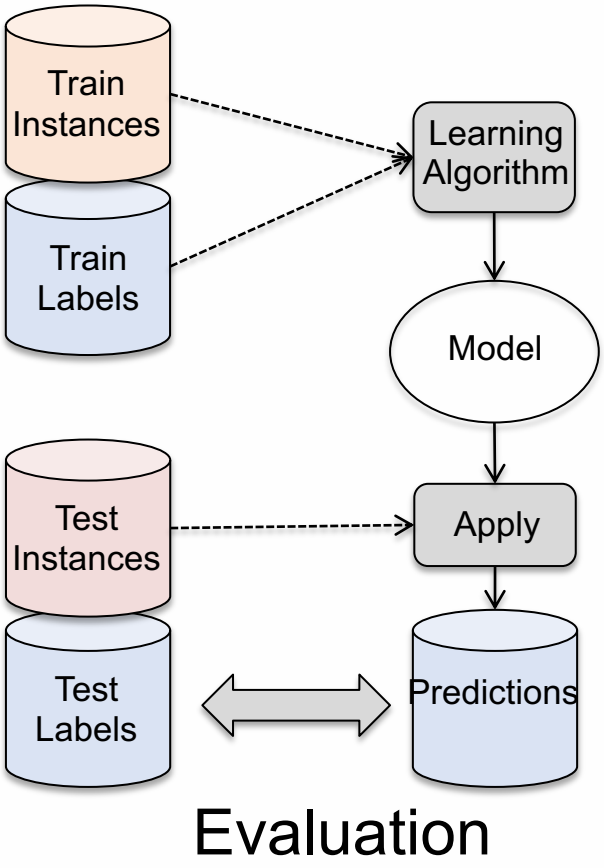
Day	Outlook	Temperature	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Training a Model

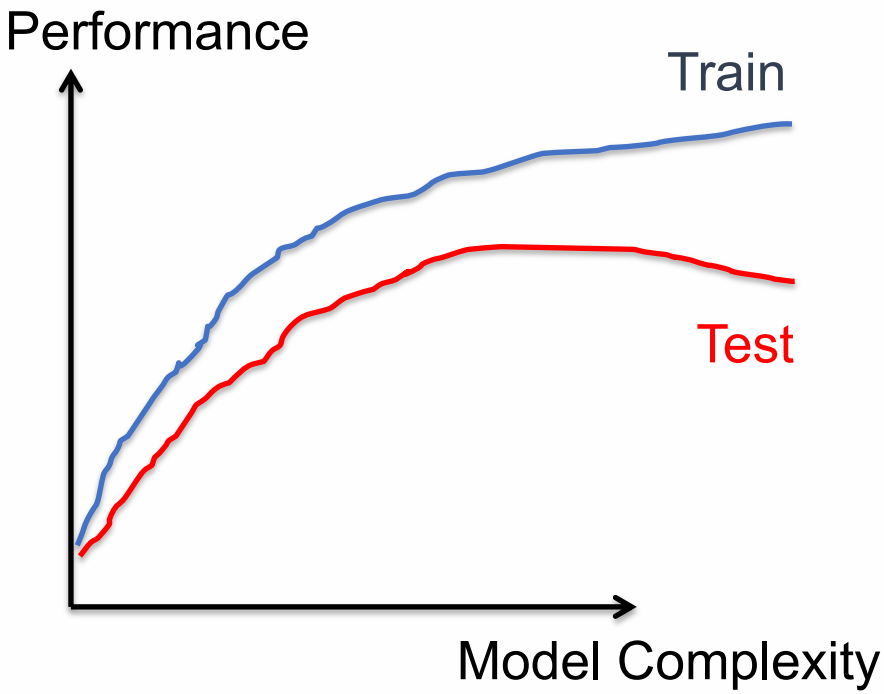


- The learning algorithm searches to find the best settings of the parameters (the coefficients) of the model
- The best settings are those that minimise error of the predictions on the training data
- Once the model has been found, it can be used to make predictions on new instances

Training and Testing



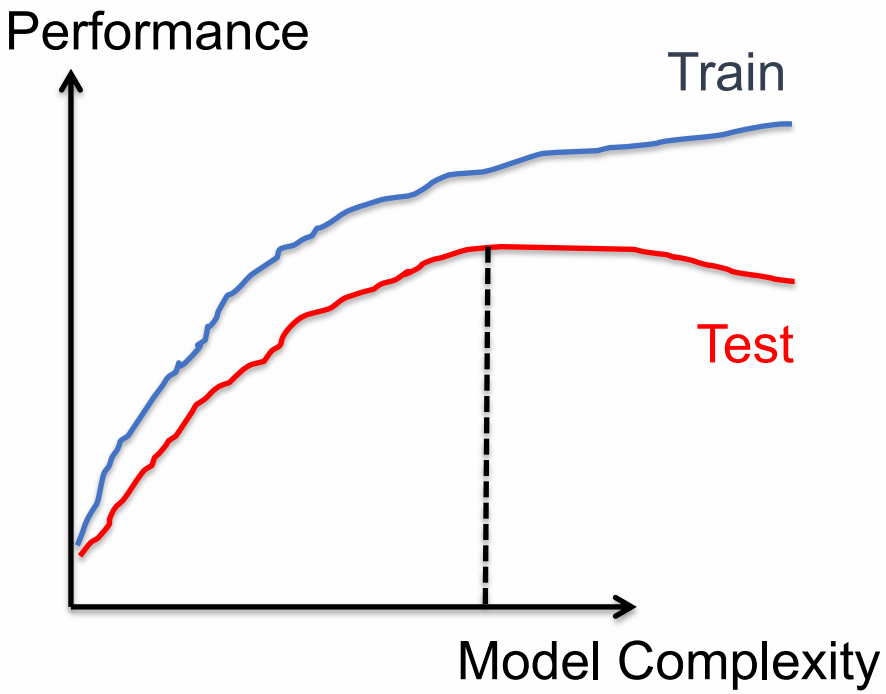
Overfitting



Training error improves with model complexity

- But test error doesn't

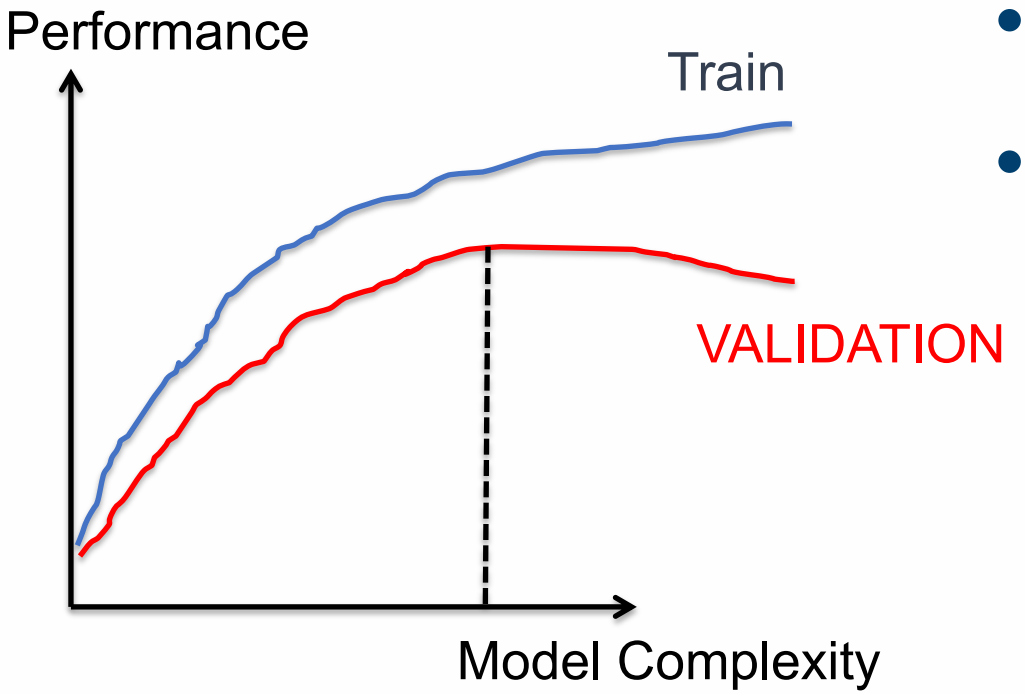
Preventing Overfitting



All (good) learners have hyper-parameters which control model complexity

- Set these hyper-parameters to prevent overfitting

Preventing Overfitting



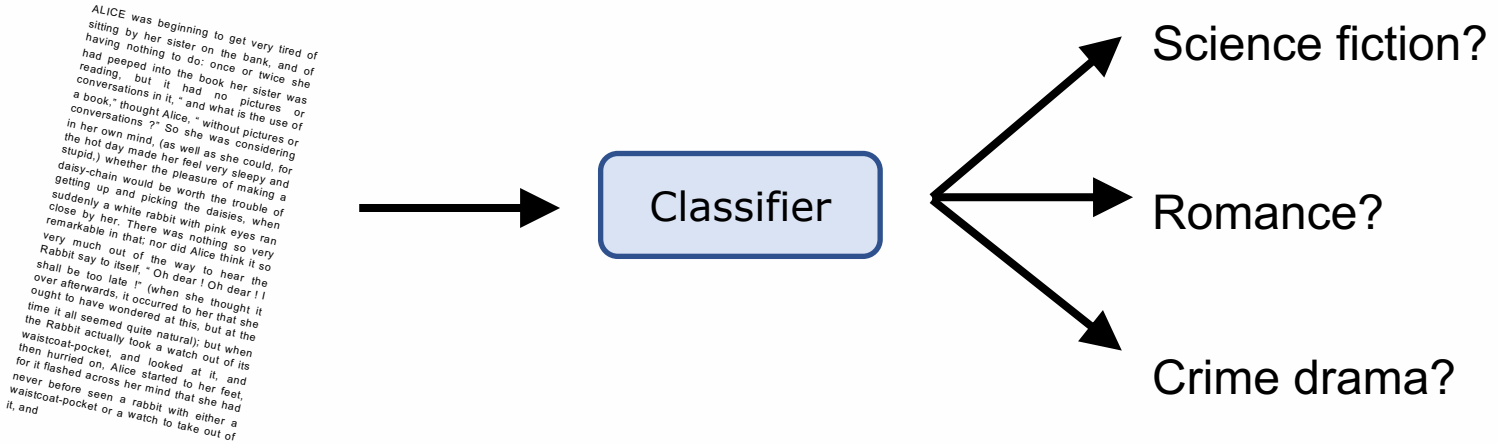
Split training data into train and validation sets

- Perform sweep over hyper-parameters
- Choose hyper-parameter value that maximises validation performance

Text Classification

What is text classification?

- Process of **training a model** to classify documents into categories



Why would I want to classify text?

Text classification is an **extremely common task**

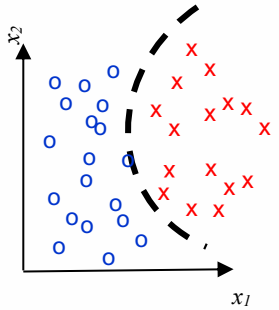
- Email spam detection
- Authorship identification
- Sentiment analysis in product reviews
- Offensive content detection
- Web search query intent identification
- Creating your news feed on Facebook/LinkedIn
- Identifying criminal behaviour online (fraud, grooming,..)
- Routing company email communications to the right person
- Parsing requests to spoken interfaces (Alexa, Siri, …)
- …

Types of text classification PROBLEMS

Binary classification problems (output is binary)

- E.g. spam detection, offensive content detection, sentiment analysis:

"I absolutely love this product. It's made my life so much better" => POSITIVE



Ordinal regression problems (output is an ordinal)

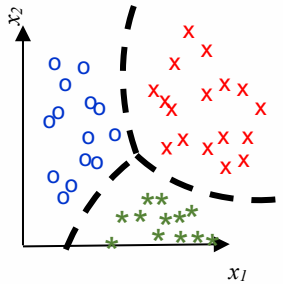
- E.g. product reviews – what is the star rating?

"Hotel room was filthy. Didn't look like it had ever been cleaned" => 1_STAR

Multi-class classification problems (output is a category)

- Categorising topic, routing communications to correct department

"Hi, My internet connection has been dodgy for the last …" => REPAIRS_DEPT



Multi-label classification problems (output is a set of categories)

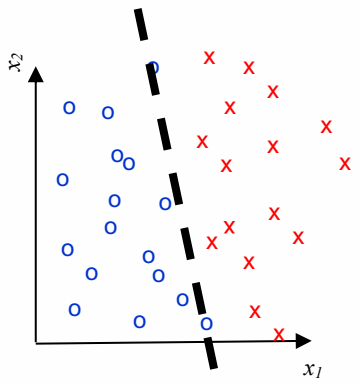
- Categorising news articles:

"Donald Trump invited Tiger Woods for a round of golf at his Florida resort"
=> POLITICS, SPORT

Types of classification MODELS

All classifiers divide up the feature space

- boundary can be linear or non-linear

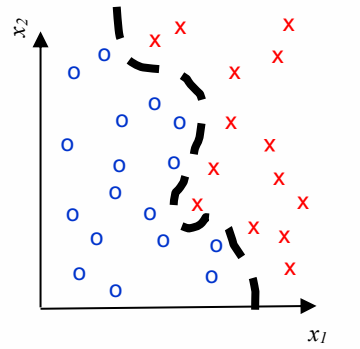


Linear models

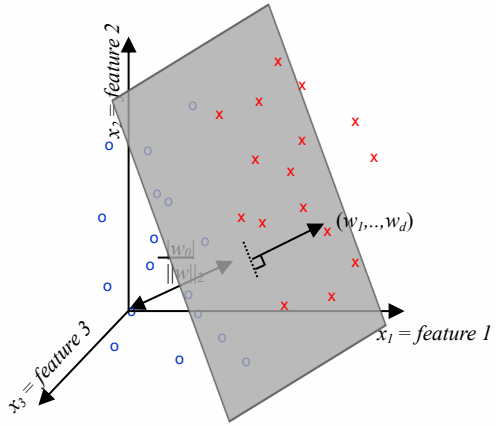
- include Naïve Bayes, Logistic Regression and Support Vector Machine (SVM)

Non-linear models

- include SVM with Radial Basis Function (RBF) kernel, Gradient Boosted Decision Trees and Neural Networks



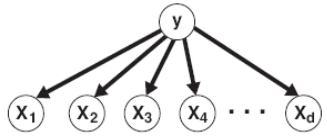
Linear Models



Find a hyperplane in d-dimensions that best separates the 2 classes

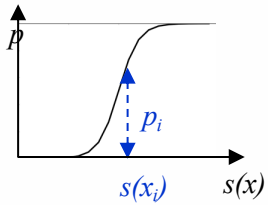
Multinomial Naïve Bayes

- makes simplifying assumption about independence of features
- traditional and very fast model to compute



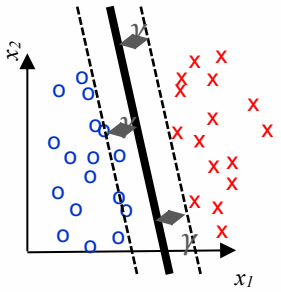
Logistic Regression

- assumes log-odds increases linearly with distance from boundary
- provides well calibrated probability estimates

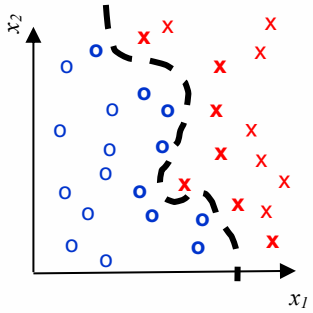


Support Vector Machine

- maximimises "buffer" either side of decision boundary
- very robust for high dimensional data

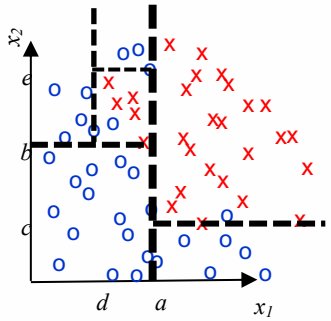


Non-linear Models



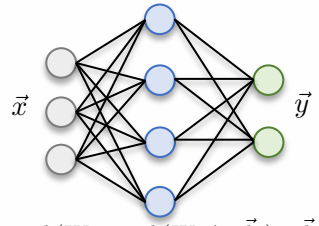
SVM with Radial Basis Function (RBF) kernel

- used to be particularly popular
- not much used with text data



Gradient Boosted Decision Trees (like XGBoost)

- very popular classifier, due of amazing performance
- but not so much with text



Neural Networks

- basic neural networks are not used for text
- but **deep recurrent neural networks** and **transformers** are massively popular

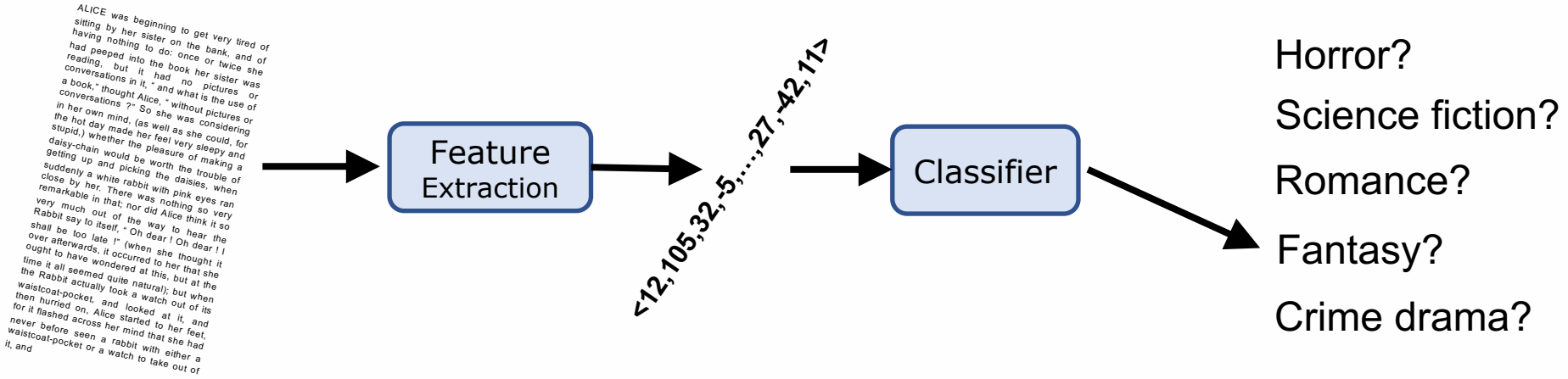
$$\vec{y} = tanh(W_2 \cdot tanh(W_1\vec{x} + \vec{b}_1) + \vec{b}_2)$$

Extracting Features from Text

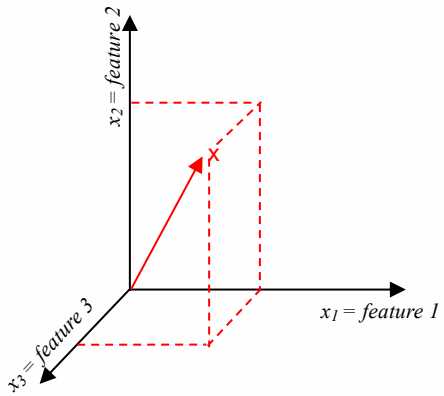
Extracting features

Text cannot be given directly to the model

- instead **features must first extracted** from the text



Extracting features from text



Features: **signals** in doc that are useful for predicting category

- need to convert text data into vector of features to give to classifier

If training data is **scarce** (few documents available) one might use:

- **Syntax-based features**, e.g. # of capitalised words
- **Part-of-speech based features**, e.g. # of verbs versus proper nouns
- **Reading-difficulty** based features, e.g. average length of words/sentences, etc.

Most common features to extract are words themselves

- vocabulary of document provides important signal
- # of occurrences of words provides further information
- **bag-of-words** model:
  - represent docs as vectors of word counts
  - massively sparse representation (long vector with many zeros)

Pre-processing text

Common to pre-process the text, performing cleaning activities, like:

- Before tokenisation:
  - **Remove** mark-up (non-content information), e.g. <HTML> tags
  - **Lowercase** the text (can lose information: e.g. 'WHO' vs 'who')
  - Remove **punctuation** (;,&%$@!><? etc.)
- After tokenisation:
  - Remove **stopwords** (extremely high frequency words)
  - Remove **low frequency words**
  - Perform **stemming** or **lemmatization**
    - e.g. fishing => fish, thought => think, etc.
  - Perform **spelling correction**

Bag-of-words representation

Motivating bag-of-words representation

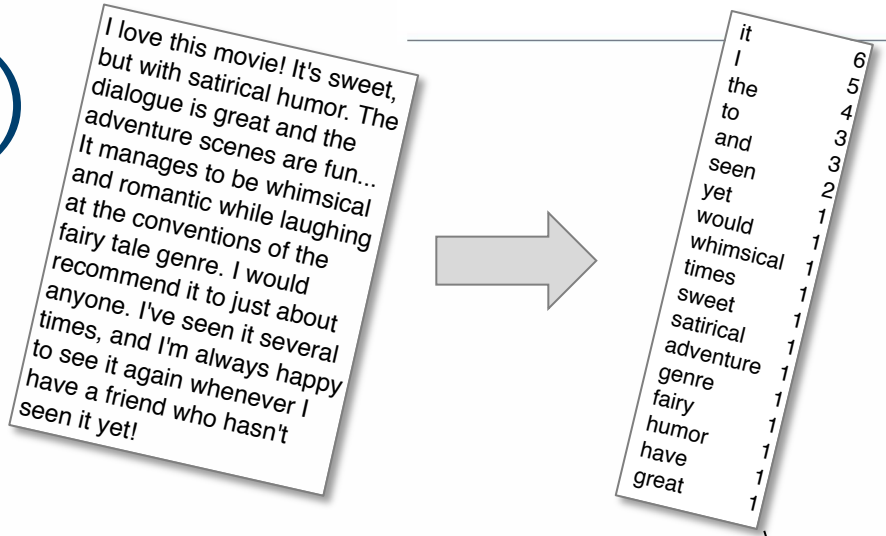
Documents are sequences of categorical variables (words from vocabulary)

- usually, in ML we **one-hot encode** categorical variables
  - produces new binary feature for every possible value of categorical variable
  - shirt-size → shirt-size-S, shirt-size-M, shirt-size-L, shirt-size-XL, shirt-size-XXL
- but text is too large
  - to one-hot encode a sequence of length 100 with a vocabulary of 20,000 would require $(20{,}000)^{100}$ columns >>> #particles in known universe
- so to reduce space, sum all one-hot encodings together
  - reduces #columns to size of vocabulary
  - **throws away** all **word order information** but retains critical vocabulary information
- consider which reviews are positive and which are negative …

… zany characters and richly applied satire, and some great plot twists	✔
It was pathetic. The worst part about it was the boxing scenes …	✘
… awesome caramel sauce and sweet toasty almonds. I love this place!	✔
… awful pizza and ridiculously overpriced …	✘

  - much of the **useful information** for classifying documents is **in the vocab**

Bag-of-words (BOW) representation



Vocabulary of document is very small compared to vocabulary of collection

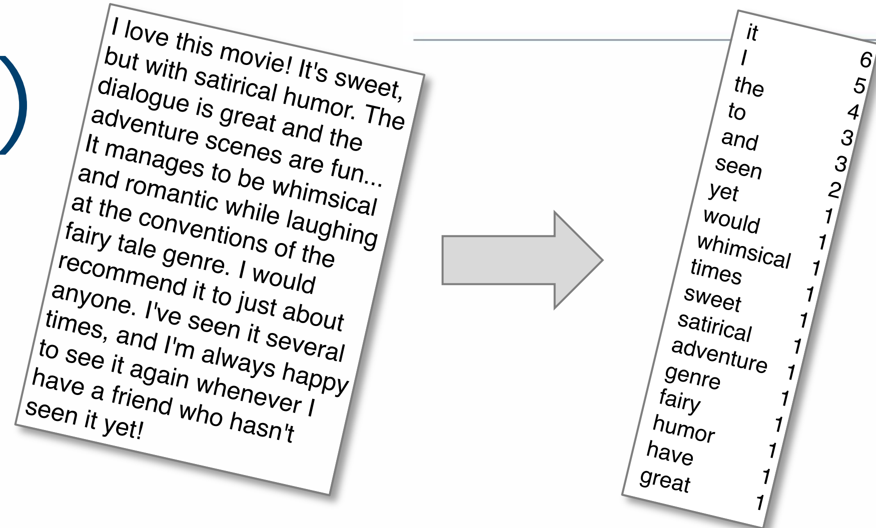
- so terms present in document usually characterise well content of document
- in BOW representation include also count of occurrences of each term
  - could use binary representation of document with little information loss
- very sparse representation:

[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, …]

Completely ignores word order:

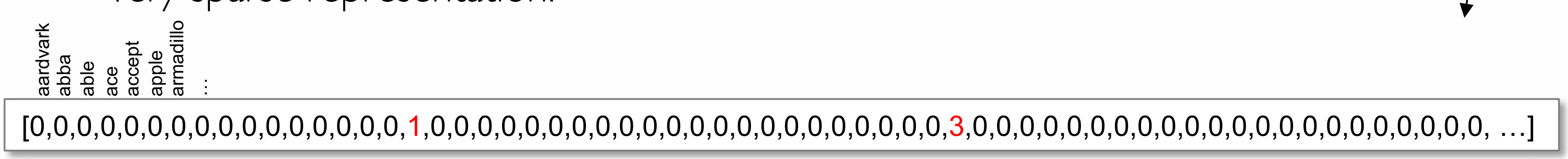
- extension to include N-grams (bigrams, trigrams) can increase performance
- but **greatly** increases number of dimensions, so more data is then needed

Bag-of-words (BOW) representation



Vocabulary of document is very small compared to vocabulary of collection

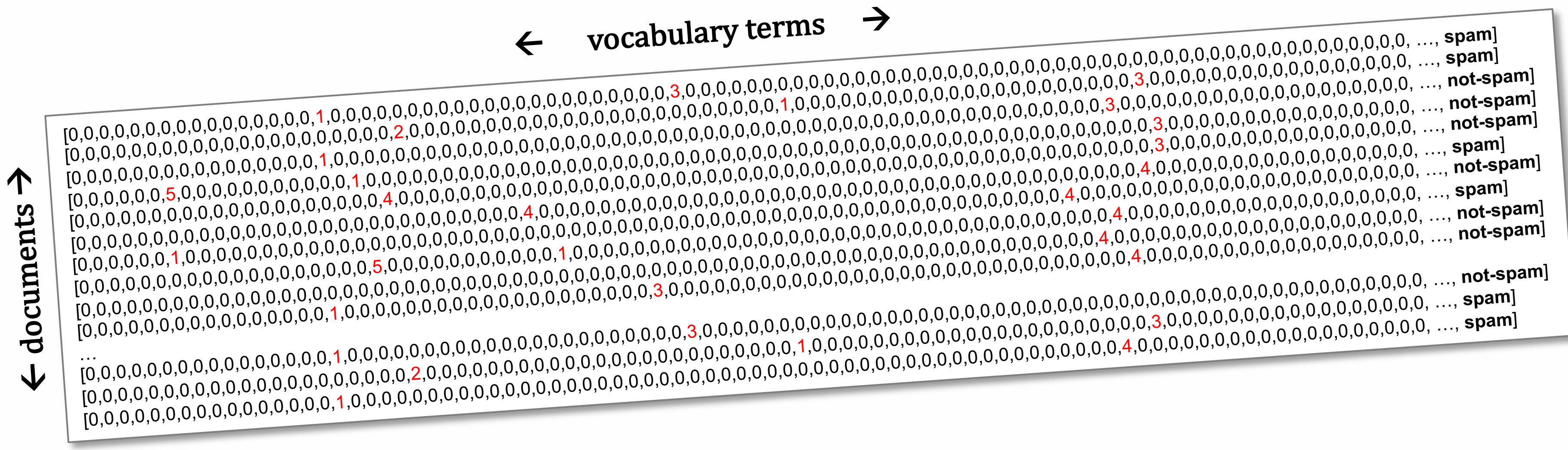
- so terms present in document usually characterise well content of document
- in BOW representation include also count of occurrences of each term
  - could use binary representation of document with little information loss
- very sparse representation:



Completely ignores word order:

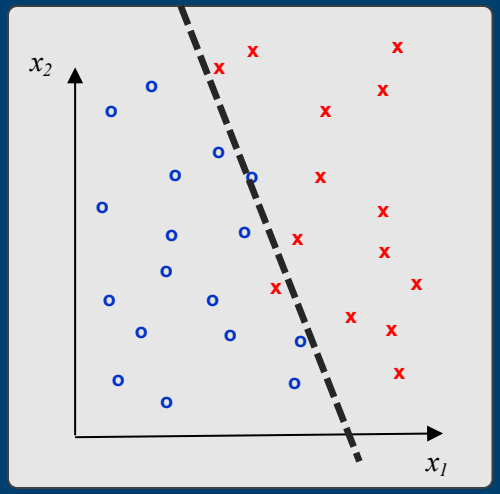
- extension to include N-grams (bigrams, trigrams) can increase performance
- but **greatly** increases number of dimensions, so more data is then needed

Bag-of-words representation



Usually have far fewer documents than vocabulary terms

- which means fewer examples than features 😱
  - this is not a "happy place" for a classifier to find itself ;-)
  - means that multiple settings of parameters can explain observed class labels
- so strong regularization is needed to guide the learner and prevent overfitting



Linear Classification Models

Some slide content based on textbook:
Machine Learning and Security: **Protecting Systems with Data and Algorithms** by Clarence Chio & David Freeman

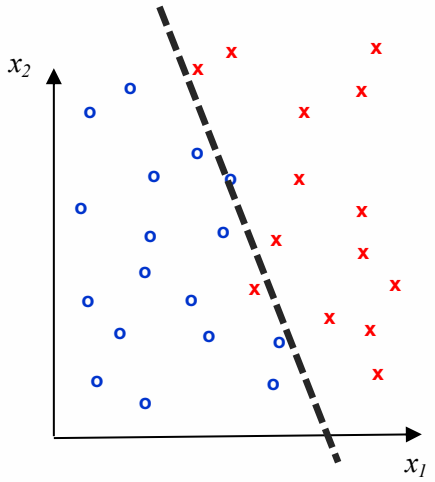
Linear Classification Algorithms

stamp corpulent popularity by geralt (Pixabay Licence)

Due to the very high number of dimensions in a bag-of-words representation of documents, linear models are often used with text

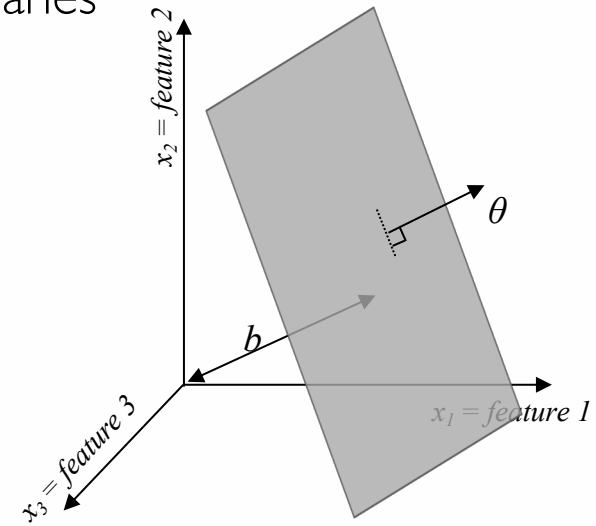
Now discuss quickly the 3 most popular **linear models**:

- Naive Bayes
- Logistic Regression
- Support Vector Machines



Decision Boundaries: hyperplanes

Linear classification algorithms find linear decision boundaries = oriented hyperplane in *n*-dimensional vector space



Given a **feature vector** $x = (x_1, \ldots, x_n)$

- and a set of ***model parameters*** denoted θ,
- the oriented hyperplane has equation:

$$\theta \cdot x - b = 0$$

- where:
  - $\theta \cdot x = \sum_i \theta_i x_i$ is a dot product
  - $\theta = (\theta_1, \ldots, \theta_n)$ is *n*-dimensional vector that is orthogonal to hyperplane
  - b is an offset, indicating distance of hyperplane from origin
    - if $|\theta|_2 = \sqrt{(\theta \cdot \theta)} = 1$ distance is b, otherwise distance is $b/|\theta|_2$
    - b is just another parameter, so often denoted $b = -\theta_0$

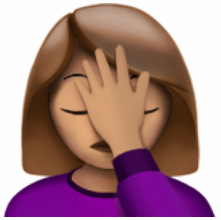
Linear Classifiers: Multinomial Naïve Bayes

Who are you calling Naïve?

Naïve Bayes (NB) is one of **oldest & simplest text classifiers**

Called naïve because it makes simplifying assumption:
that word occurrences are ***statistically independent*** *of each other given the* ***class label***

- i.e. words provide **independent information** about the class
- assumption makes calculating parameters of model very simple (see next slide)
- assumption **does not hold in practice, since words are highly correlated with each other**
- **but nonetheless the predictions from the model are good**
  - The assumption just causes model to be overconfident



Multinomial Naïve Bayes Classifier

Want to estimate the probability of spam given text content of email:

$$P(\text{spam} \mid \textit{hi mark I am a student in your nlp course and } \ldots)$$

- can't estimate class probability directly, because haven't seen any emails **exactly** like it before (with same set of words)
- so use Bayes' rule to swap the order of the events:

$$= \frac{P(\textit{hi mark I am a student in your nlp course and } \ldots \mid \text{spam})P(\text{spam})}{P(\text{“}\textit{hi mark I am a student in your nlp course and } \ldots\text{”})}$$

- ignore denominator, since it's the same for both spam / not spam, and we can normalize later…

$$\propto P(\textit{hi mark I am a student in your nlp course and } \ldots \mid \text{spam})P(\text{spam})$$

- now make **simplifying** (and clearly wrong) **assumption** that words occurrences are not correlated within the classes
  - i.e. assume $P(\textit{hi} \mid \textit{mark I am a student in your nlp course and} \ldots, \text{spam}) = P(\textit{hi} \mid \text{spam})$

$$\propto P(\textit{hi} \mid \text{spam})P(\textit{mark} \mid \text{spam})P(\textit{I} \mid \text{spam})P(\textit{am} \mid \text{spam})P(\textit{a} \mid \text{spam}) \ldots P(\text{spam})$$

- Finally **estimate probabilities** from training data:

$$P(\textit{mark} \mid \text{spam}) \approx \frac{\text{number of spam emails containing word ‘}\textit{mark}\text{’}}{\text{number of spam emails}}$$

$$P(\text{spam}) \approx \frac{\text{number of spam emails}}{\text{number of all emails}}$$

NB – smoothing probability estimates

What happens if all examples of a word occur for only one class?

- e.g. word 'inheritance' appears only in spam messages?
  - estimate *P(inheritance*|**not-spam**) = *0*
  - so probability for **not-spam** class will be zero,
  - doesn't make sense since we may just have not seen enough data yet…

Avoid problem by **smoothing** the probability estimates

- add pseudo-count α for each feature as follows:

$$P(\textit{inheritance} \mid \textbf{not-spam}) = \frac{\#\text{ not-spam emails containing word '}\textit{inheritance}\text{'} + \boldsymbol{\alpha}}{\text{number of not-spam emails} + 2\boldsymbol{\alpha}}$$

  - α can be chosen to maximise prediction performance or set to default value (if $\alpha = 1$ technique is called *Laplace smoothing)*

NB – Out of Vocabulary

Hi again,
I have an **amaaaaaaaaaazing** one-time offer for you.
Click on this link below to find out more:
http://a-very-dangerous-link.com/do-not-click
Your friend,
The Spammer

What happens if a new word appears in test set that wasn't in training set?

- we have no estimate for *P(amaaaaaaaaaazing*|spam*)*, so classifier just ignores that feature …

How is NB a linear classifier?

Naïve Bayes computes product of conditional probabilities of words in document:

$$P(\text{spam} \mid \text{document}) \propto [\prod_{positions} P(\text{word}_i \mid \text{spam})]\, P(\text{spam})$$

- Considering both classes together, we can calculate the odds of spam:

$$\left[\frac{P(\text{spam} \mid \text{document})}{P(\text{not-spam} \mid \text{document})}\right] = \left[\prod_{\text{positions}} \frac{P(\text{word}_i \mid \text{spam})}{P(\text{word}_i \mid \text{not-spam})}\right]\left[\frac{P(\text{spam})}{P(\text{not-spam})}\right]$$

- Taking the log will convert all the multiplications to additions:

$$\log\left[\frac{P(\text{spam} \mid \text{document})}{P(\text{not-spam} \mid \text{document})}\right] = \sum_{\text{positions}} \log\left[\frac{P(\text{word}_i \mid \text{spam})}{P(\text{word}_i \mid \text{not-spam})}\right] + \log\left[\frac{P(\text{spam})}{P(\text{not-spam})}\right]$$

- And summing over vocabulary (features) rather than positions within the document then gives us a linear equation of the document feature vector x_i:

$$\log\left[\frac{P(\text{spam} \mid \text{document})}{P(\text{not-spam} \mid \text{document})}\right] = \sum_{\text{vocab}} \log\left[\frac{P(\text{word}_i \mid \text{spam})}{P(\text{word}_i \mid \text{not-spam})}\right] x_i + \log\left[\frac{P(\text{spam})}{P(\text{not-spam})}\right]$$

- This is the standard linear equation for a linear classifier: $\mathbf{f}(x) = \sum_i \theta_i x_i + \theta_0$
  - classify as spam if positive, no-spam if negative

Is the independence assumption a big deal? Not really …

Spam classifier uses presence of words as features:

- independence assumption posits that:
  - knowing whether spam email contains word *"cheap"* ***doesn't change probability*** of seeing word *"save"*
  - expressed in terms of conditional probabilities: $P(\text{'save'} \mid \textbf{spam}) = P(\text{'save'} \mid \text{'cheap'}, \textbf{spam})$
- assumption obviously incorrect since words are **correlated** with each other
- but simplifies problem of estimating model and making predictions:
  - see word '**cheap**' in email => **very likely spam**, increase ***spam score*** by 3
  - see word '**save**' in email => **likely spam**, increase ***spam score*** by 2
  - see both 'cheap' & 'save' in email => increase ***spam score*** by 3 + 2 = 5
  - in theory should **increase score by less** because terms are not independent
- in practice, NB learns effective spam detector

NB – pros & cons

Advantages of NB:

- very **fast** to estimate NB model
  - requires only one pass over the training data!
  - no need to search for parameters (no gradient descent routine)
- reliable predictor even if there is little data available (stable classifier)
  - if conditional independence assumption holds, it's the best

Disadvantages of NB:

- doesn't tend to perform quite as well on large data as other classifiers
  - especially if many redundant features since information is being counted twice!
- predicted probabilities are not well calibrated
  - predictions are often **overconfident** due to violations of independence assumption

Classification Algorithms: Logistic Regression

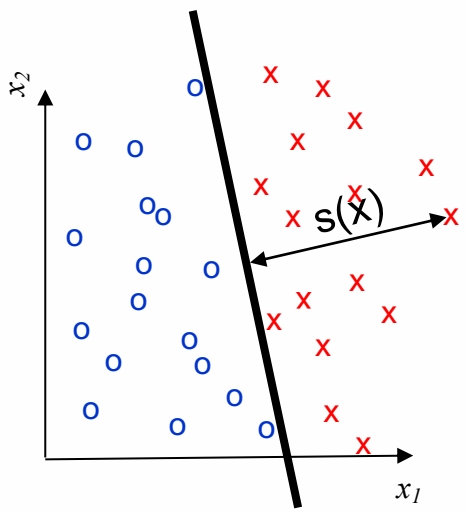
Distance from Hyperplane

The further a point is from decision boundary

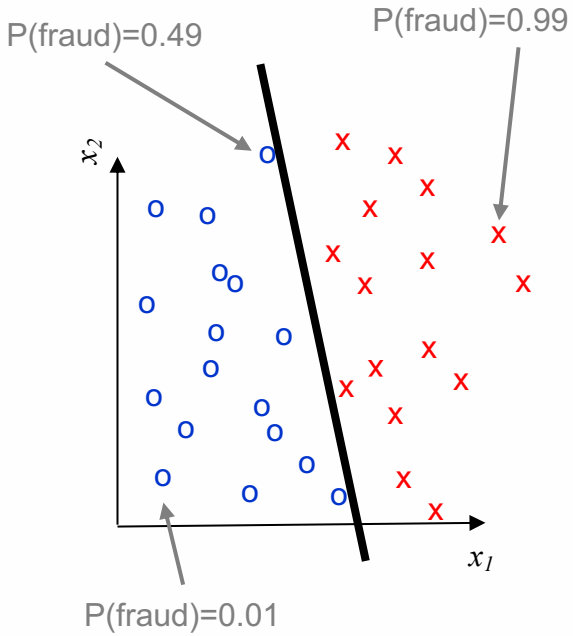
- the more certain we are about the prediction

Signed distance of a point from hyperplane is: $s(x) = \theta \cdot x - b$

- points with:
  - **positive** score are likely **spam**
  - **negative** score are likely **not-spam**



From Distances to Probabilities



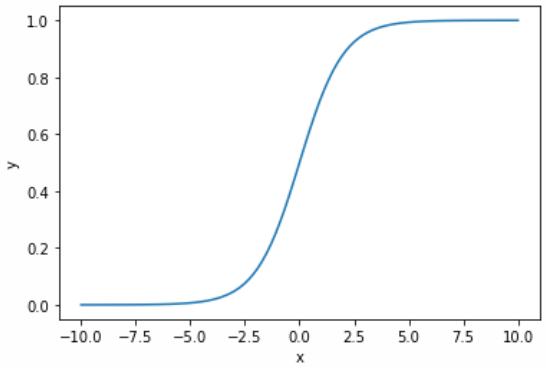
Instead of mapping each point to *fraud/non-fraud*

- could try to ***estimate the probability*** of fraud
  - interpreted as the **confidence of the classification**

To convert distance $s(x)$ to probability $P(fraud \mid x)$:

- need function that maps $(-\infty,+\infty) \rightarrow [0,1]$
- standard function is ***logistic curve*** (aka ***sigmoid function***)

$$sigmoid(s) = \frac{1}{1+e^{-s}} = \frac{e^{s}}{e^{s}+1}$$



  - takes value 0.5 at decision boundary (s=0)
  - slope (speed of probability change) depends on size of θ

What is Logistic Regression?

The Betting for the Bumper by Will Palmer (CC BY 2.0)

Logistic regression (LR)

- **very** popular algorithm for learning a statistical classifier
- requires numerical features
- assumes ***log-odds*** of class label is linear function of feature values

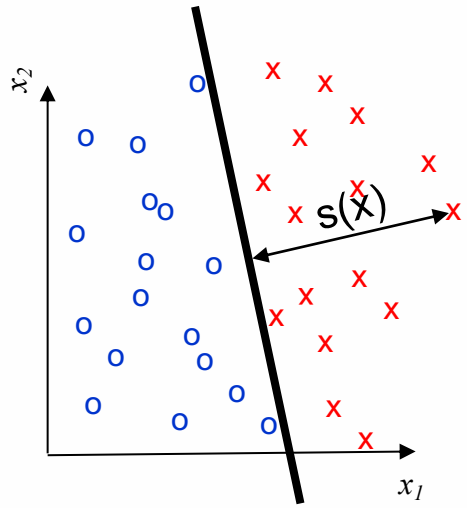
What are the *log-odds*?

- ***odds*** of event is ratio of chance it happens vs doesn't: $\frac{P(\text{fraud}|x_i)}{P(\neg\text{fraud}|x_i)} = \frac{p_i}{1-p_i}$
  - odds gives *multiplying factor:* **how much more/less likely** is event to happen than not happen?
- taking *logarithm of the odds*, gives an *additive factor*
  - difference between log probability of event happening or not happening

examples:
- odds of **tossing a head** is 1/1
- odds of **rolling a 6** is 1/5

$$\log(\frac{P(\text{fraud}|x_i)}{P(\neg\text{fraud}|x_i)}) = \log(P(\text{fraud}|x_i)) - \log(P(\neg\text{fraud}|x_i))$$

So why is it called *Logistic* Regression?



LR uses linear model (distance from hyperplane) to predict the log-odds:

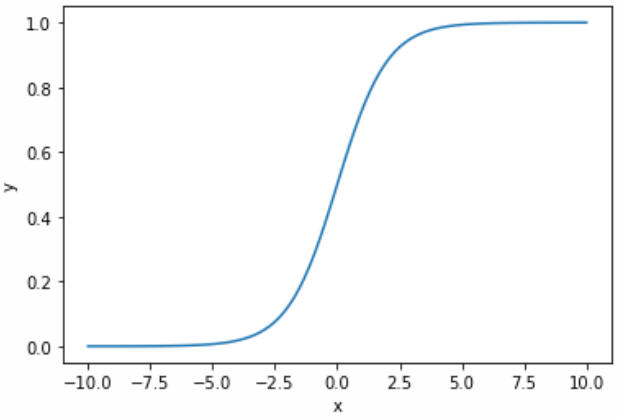
$$s(x_i) = \theta \cdot x_i + b \approx \log(\frac{p_i}{1-p_i})$$

- if we invert the log odds function:

$$s(x_i) = \log(\frac{p_i}{1-p_i})$$

$$\frac{p_i}{1-p_i} = e^{s(x_i)}$$

$$p_i = \frac{e^{s(x_i)}}{1+e^{s(x_i)}} = \frac{1}{1+e^{-s(x_i)}}$$



- we get the **logistic** curve (aka sigmoid function)

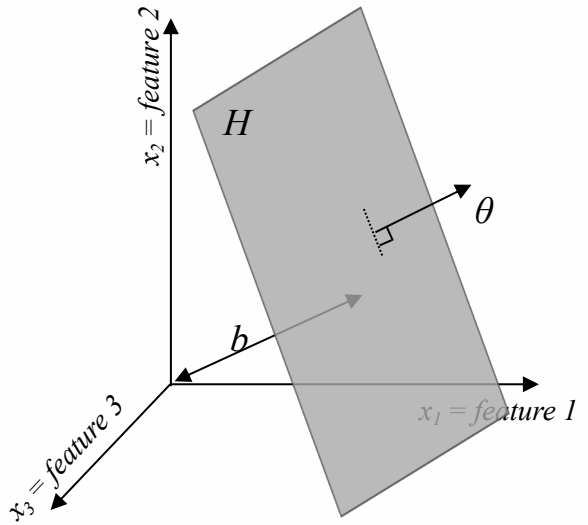
Logistic Regression: decision boundary

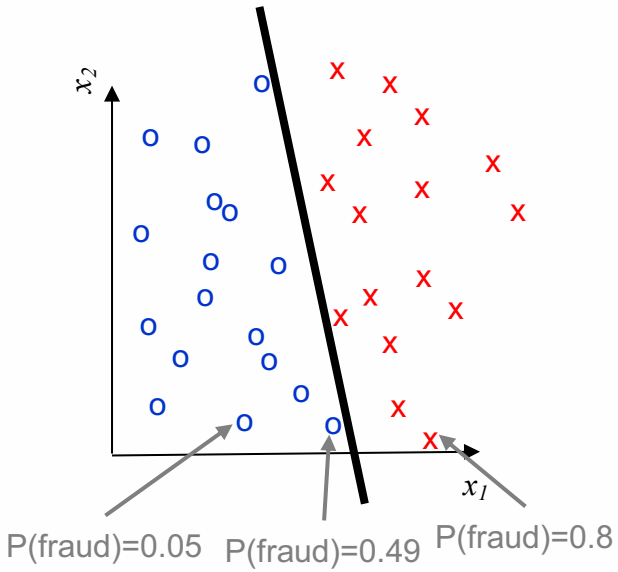
Log-odds of predicted probability is linear function

$$\log(\frac{P(\text{fraud}|x_i)}{P(\neg\text{fraud}|x_i)}) = \theta \cdot x_i + b$$

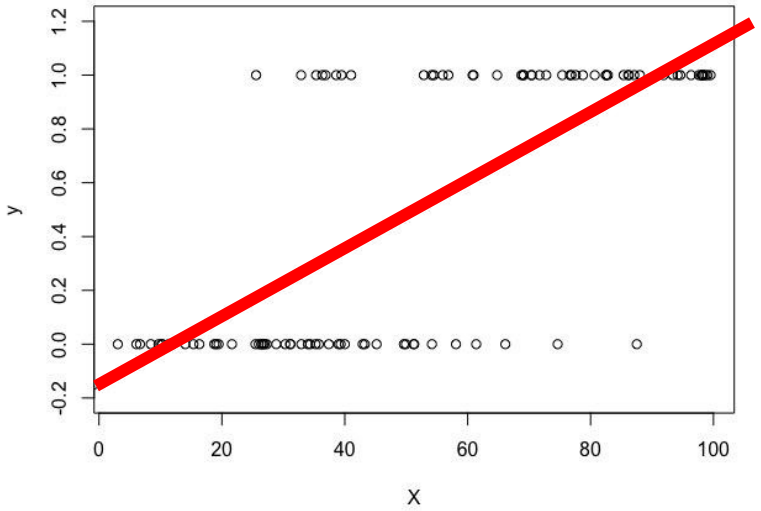
- when probability is 0.5, log-odds is 0
  - classify point as positive if log-odds > 0
  - so LR learns a linear decision boundary

- changing value of a feature
  - increases or decreases predicted probability monotonically
  - depending on whether coefficient is positive or negative





Why logistic rather than linear regression?



Why not use just use linear regression for building classifier?

- linear regression predicts *real-valued* y as linear combination of feature vector x

  $y \simeq \mathrm{f(x)} = \theta \bullet x + b$
- to predict binary **class label** just map categories to 0 and 1
- and compute line of best fit through them
- advantage:
  - parameters of linear regression model easier to estimate than LR

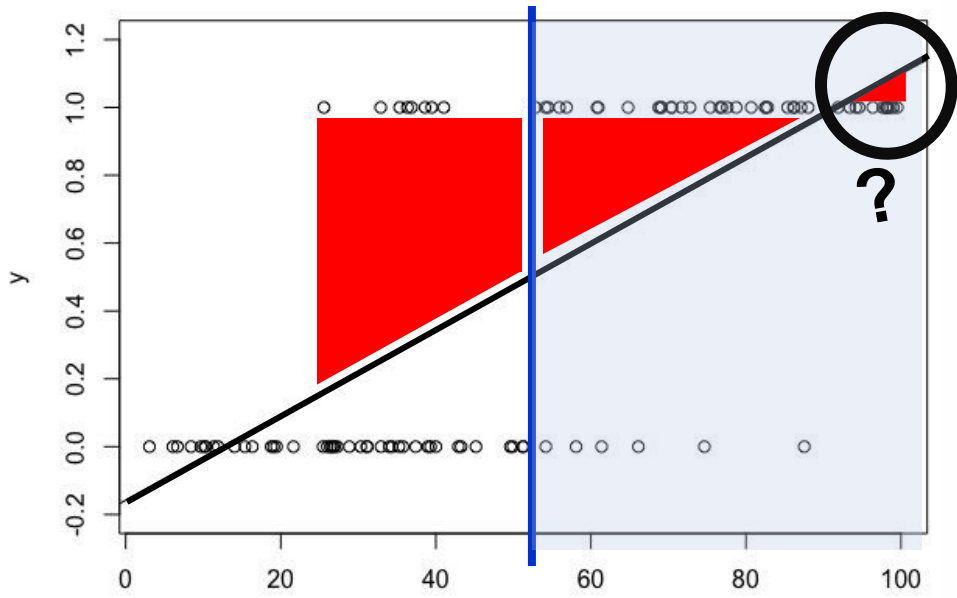
Linear vs Logistic Regression

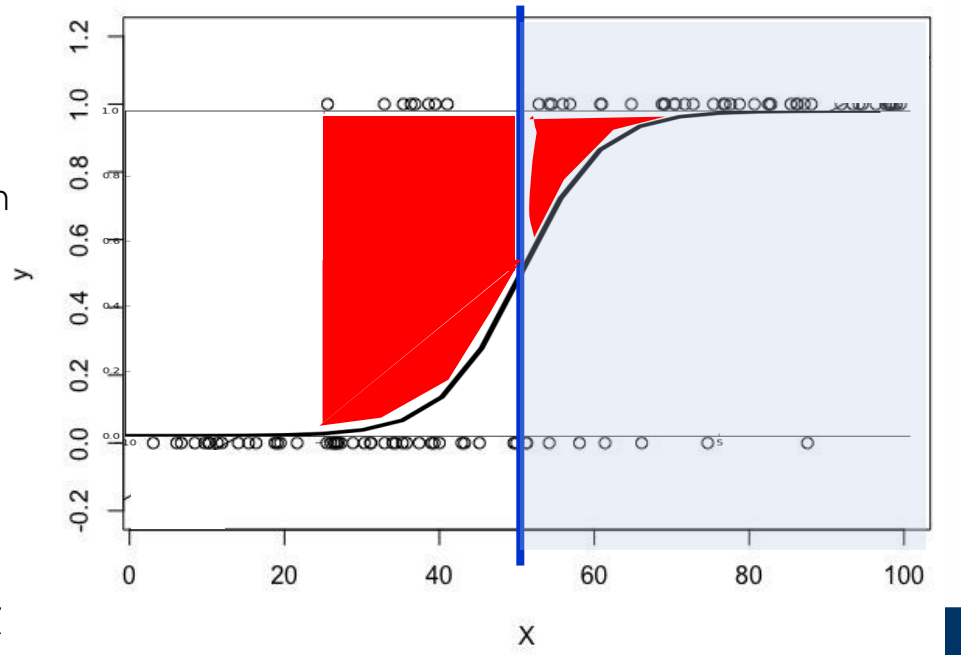
Logistic curve fits well through binary data

- results in better decision boundary!

Red area indicates size of residual error for positive examples

- to right of boundary (correct prediction):
  - linear regression error not monotone (starts increasing again to the right)
  - logistic regression error decreases monotonically to the right
- to left of boundary (incorrect pred.):
  - SSE increases quadratically with distance from hyperplane
  - for logistic regression the error ln(1+exp(-dist)) increases linearly
  - means that Logistic Regression's decision boundary is less sensitive to outliers





Logistic Regression: pros & cons

Pros:

- produces **well-calibrated probability estimates**
- can be trained efficiently and scales well to large numbers of features
- explainable since each feature's contribution to final score is additive

Cons:

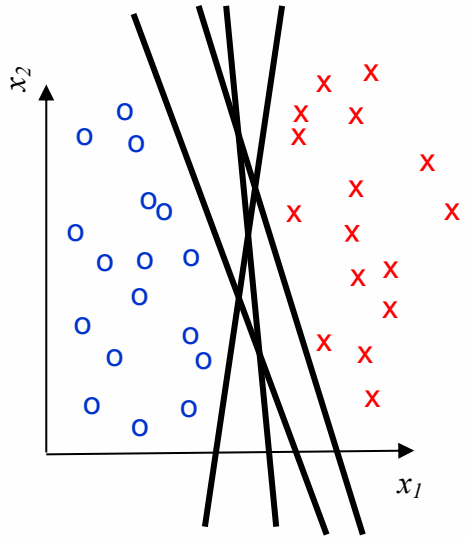
- assumes feature values are ***linearly related*** to *log odds*
  - if assumption is *strongly* violated, model will perform poorly

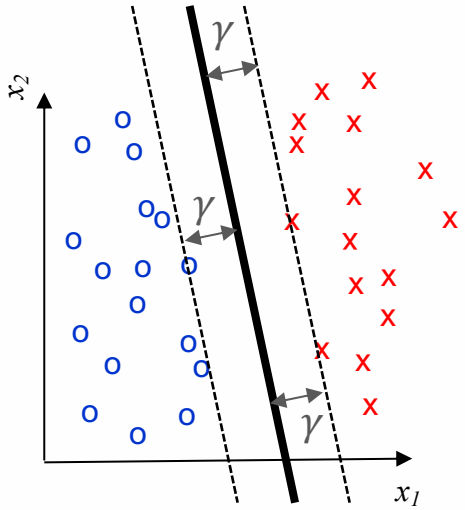
Classification Algorithms: Support Vector Machines

Maximising the margin

Imagine data for 2 classes splits nicely into 2 groups:

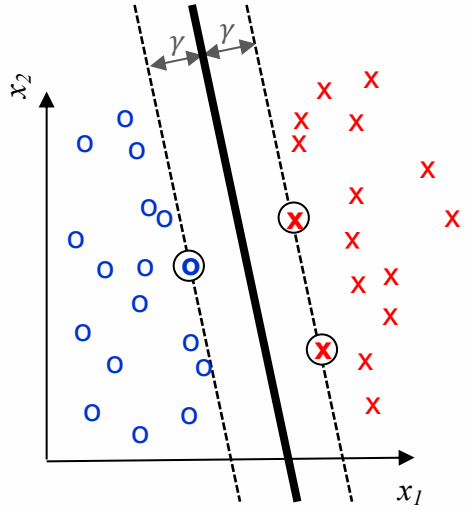
- lots of options for where to place linear decision boundary
  - which one should we choose?





- choose to *approximately* average over them
  - should be robust and generalise well to new data
- SVM finds ***maximum margin hyperplane*** separating classes
  - margin, γ = distance from hyperplane to closest points either side

Support vectors



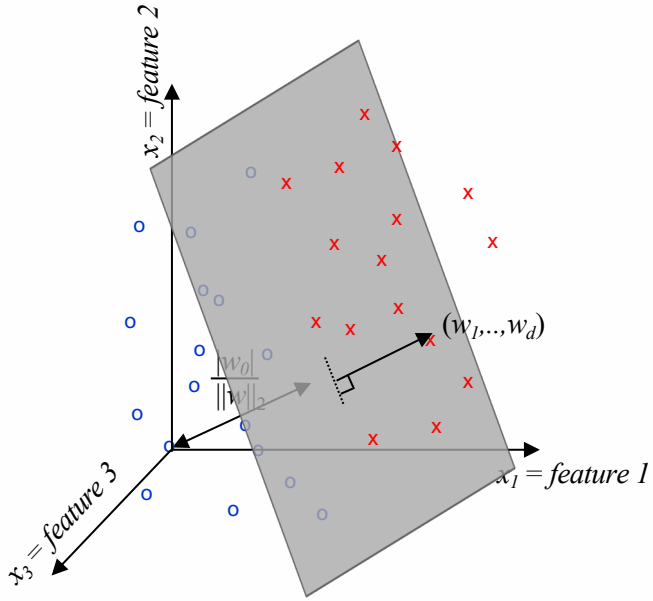
Points lying exactly on margin referred to as **support vectors**

- prevent the margin from getting bigger
- thus constrain/define location of boundary
- in d-dimensional space, have at least $d+1$ support vectors

Note difference with Logistic Regression:

- position of hyperplane depends only on closest points
  - convex hull of data points
- adding or moving internal points won't effect boundary

Support Vector Machines (SVMs)



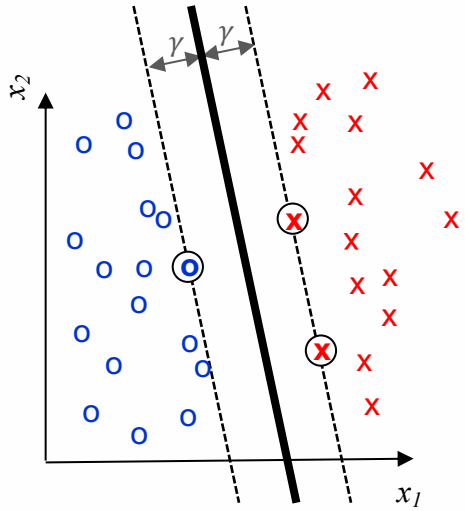
So basic *support vector machine* (SVM) is also linear classifier

- finds hyperplane in feature space that best separates the two classes

But LR and also NB also found linear decision boundaries?

- difference lies in **loss function** used to find parameters:
  - LR uses ***negative log-likelihood***: penalizes points proportionally to probability of incorrect prediction (including those for which prediction was correct)
  - SVM uses ***hinge loss***, which only penalizes points that are on the wrong side (or very close to) the hyperplane

Maximising the margins

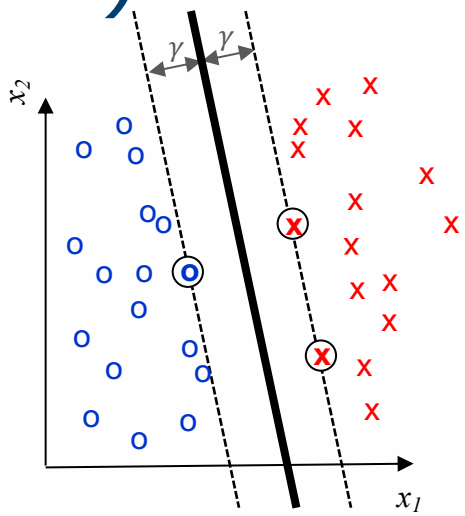


We can compute the distance to points from the hyperplane using:

- $s(x) = w_0 + \sum_i w_i x_i$

- for points that lie exactly on the margin we have:
  - $|s(x)| = |w_0 + \sum_i w_i x_i| = \gamma$
- assuming for simplicity that hyperplane passes through origin ($w_0 = 0$), then points on margin have $|\boldsymbol{w}\cdot\boldsymbol{x}| = \gamma$ where: $\boldsymbol{w}=(w_1, \ldots, w_d)$, $\boldsymbol{x}=(x_1, \ldots, x_d)$
- more generally we have:
  - $\boldsymbol{w}\cdot\boldsymbol{x} \geq \gamma$ for all positive data points, and $\boldsymbol{w}\cdot\boldsymbol{x} \leq -\gamma$ for all negative datapoints
  - equivalently, we have $y_j \boldsymbol{w}\cdot\boldsymbol{x}_j \geq \gamma$ for all examples $\boldsymbol{x}_j$ where class $y_i \in \{+1, -1\}$

Maximising the margins (cont.)



So just need to find parameters $\boldsymbol{w}=(w_1,\ldots,w_d)$

- that maximise the margin γ
- subject to the constraint: $\forall_j\ y_i\boldsymbol{w}\cdot\boldsymbol{x}_j \geq \gamma$

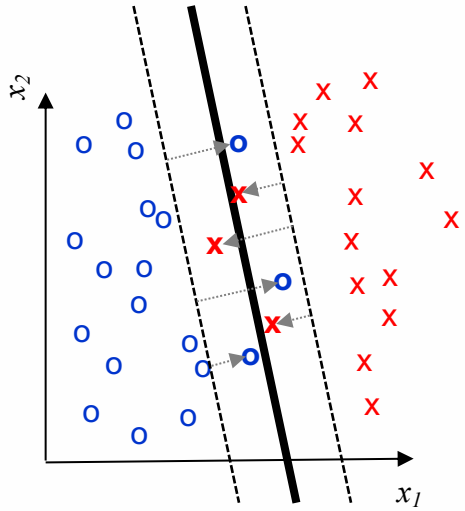
But increasing length of vector $\boldsymbol{w}$ increases value γ, so either:

- fix length of vector $\boldsymbol{w}$ (e.g. $\sum_i w_i^2 = 1$) and maximise γ
- or set $\gamma=1$ and minimize $\sum_i w_i^2$ subject to constraint: $\forall_j\ y_j\boldsymbol{w}\cdot\boldsymbol{x}_j \geq 1$

Mathematically, same as minimizing loss: $\sum_i w_i^2 + \sum_j \varepsilon_j$

- where distance of point on wrong side of margin $\varepsilon_j = \max(0, 1 - y_j\boldsymbol{w}\cdot\boldsymbol{x}_j)$ is the "error" for prediction $(\boldsymbol{x}_j, y_j)$

Non-separable data



What if data is **not linearly separable**?

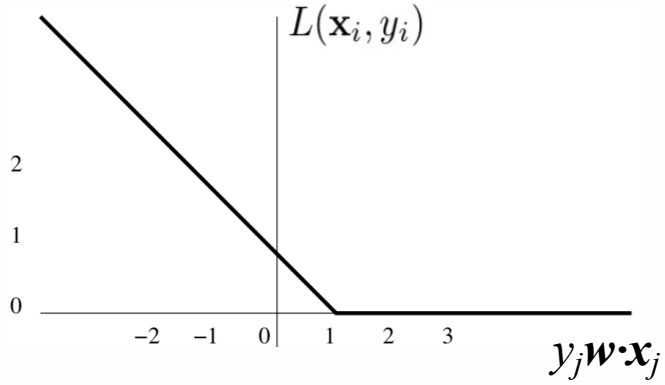
- simply penalise points on wrong side of **margin** based on distance from it
- *support vectors:* points on wrong side of margin that provide non-zero contribution to loss function
- objective function to minimise remains almost unchanged:

$$O(\vec{w}, b) = \frac{1}{2}\sum_{j=1}^{d} w_j^2 + C\sum_{i=1}^{n} \epsilon_i$$

$$= \frac{1}{2}\sum_{j=1}^{d} w_j^2 + C\sum_{i=1}^{n} \max[0, 1 - y_i(\sum_{j=1}^{d} w_j x_{ij} - b)]$$

- ϵ_i is distance from i^{th} support vector to margin
- C is a new hyperparameter that determines relative influence of errors vs size of margin

Hinge loss



Reordering objective (and dividing by C) gives more natural form:

- error over training data plus **regularisation term** that penalises complicated models
- error function is **hinge loss**
  - applies no cost to most correctly classified points
  - increases linearly with distance from margin

$$O(\vec{w}, b) = \sum_{i=1}^{n} \epsilon_i + \frac{1}{2C} \sum_{j=1}^{d} w_j^2$$

$$L(\mathbf{x}_i, y_i) = \max[0, 1 - y_i(\sum_{j=1}^{d} w_j x_{ij} - b)]$$

Summary: linear SVM

Basic *support vector machine* (SVM) is linear classifier

- finds hyperplane in vector space that separates two classes
- as does logistic regression (LR)

Difference between LR and SVMs lies in loss function:

- LR uses *log-likelihood*, which penalizes:
  - points proportionally to probability of incorrect prediction
  - including those *on correct side of hyperplane*
- SVM uses ***hinge loss***, which penalizes:
  - points linearly with distance from the hyperplane
  - that are on wrong side of hyperplane or **very close to it**

Conclusions on Linear Classifiers

Conclusions

We've seen 3 **linear** classification techniques:

- Naive Bayes
- Logistic Regression
- Support Vector Machines

Each classifier has its pros and cons:

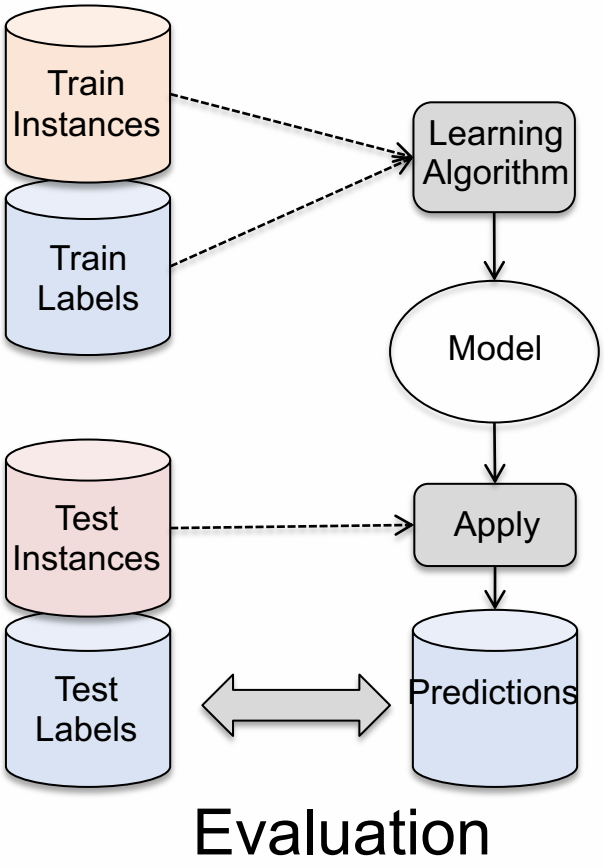
- which model will work best on a problem will depend on the text data itself
- so usually we try different classifiers and see which performs best …

If using a bag-of-words representation of text:

- Linear classifiers are sufficient for most problems
- data is very high dimensional, so **dependencies** between features don't usually need to be modelled

Evaluating Classifiers

Training and Testing



Evaluating a Binary Text Classifier

- Investigate the Confusion Matrix

- Accuracy
  = % of correct predictions
- Precision
  = % of positive predictions that were correct
- Recall
  = % of positive instances that were found by model
- F-measure
  = harmonic mean of precision and recall
  - Also AuC is often used as a single measure

		PREDICTED CLASS: Pos	PREDICTED CLASS: Neg
TRUE CLASS: Pos		TP: true positives	FN: false negatives
TRUE CLASS: Neg		FP: false positives	TN: true negatives

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

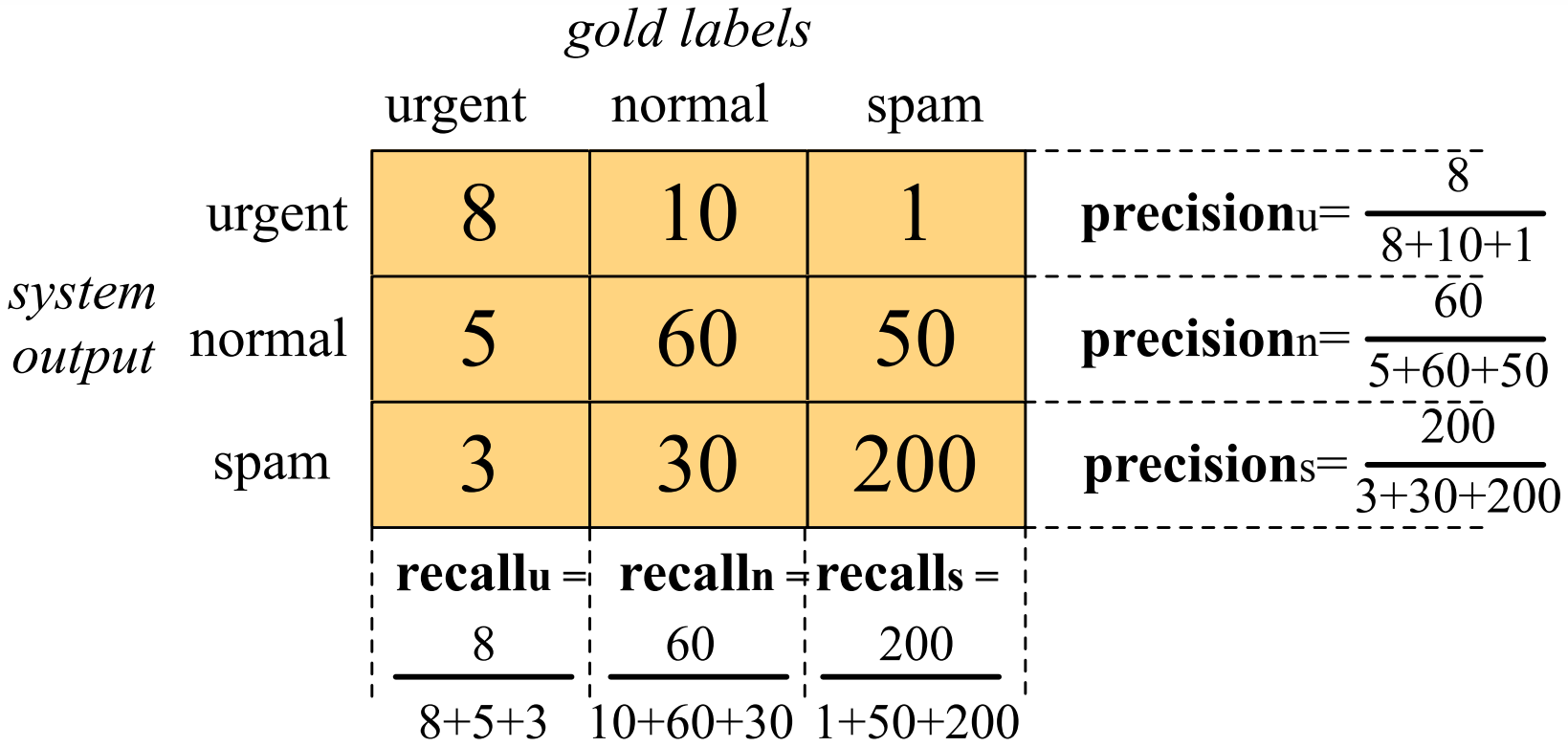
$$Precision = \frac{TP}{TP + FP}$$

$$Recall = TPR = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

What if there are more than 2 classes?

Confusion Matrix for 3-class classification



Combine P/R from 3 classes to get one metric

- Macroaveraging:
  - compute the performance for each class, and then average over classes
- Microaveraging:
  - collect decisions for all classes into one confusion matrix
  - compute precision and recall from that table.

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$

Conclusions on Text Classification

Conclusions on Text Classification

- TODO