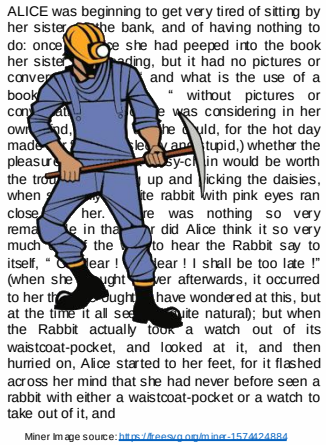
POLITECNICO DI MILANO

Classifying Text

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*



Miner Image source: https://freesvg.org/miner-1574424884

Contents:

- Quick Revision: Supervised Machine Learning
- Text Classification
- Extracting Features from Text
- Linear Models
- Evaluating a Text Classifier
- Conclusions

Quick Revision:

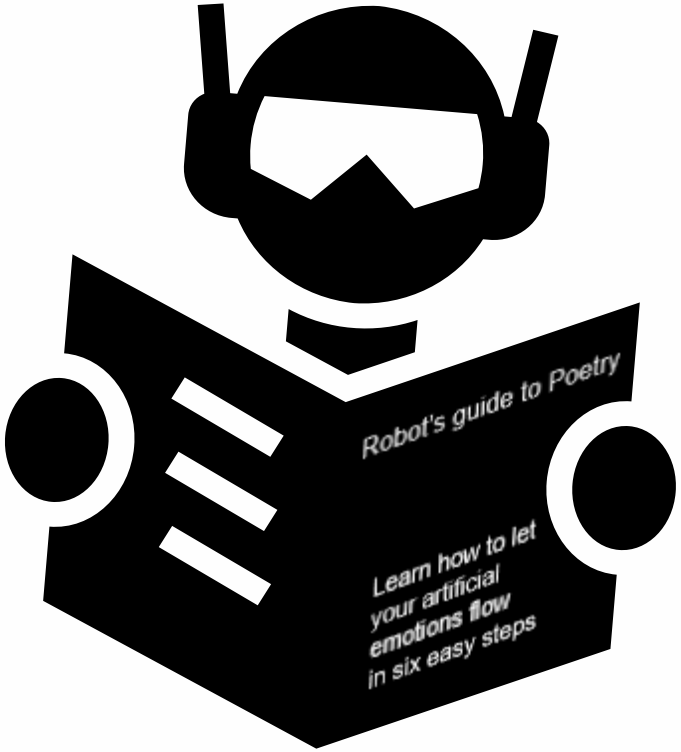
- Supervised Machine Learning

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*."



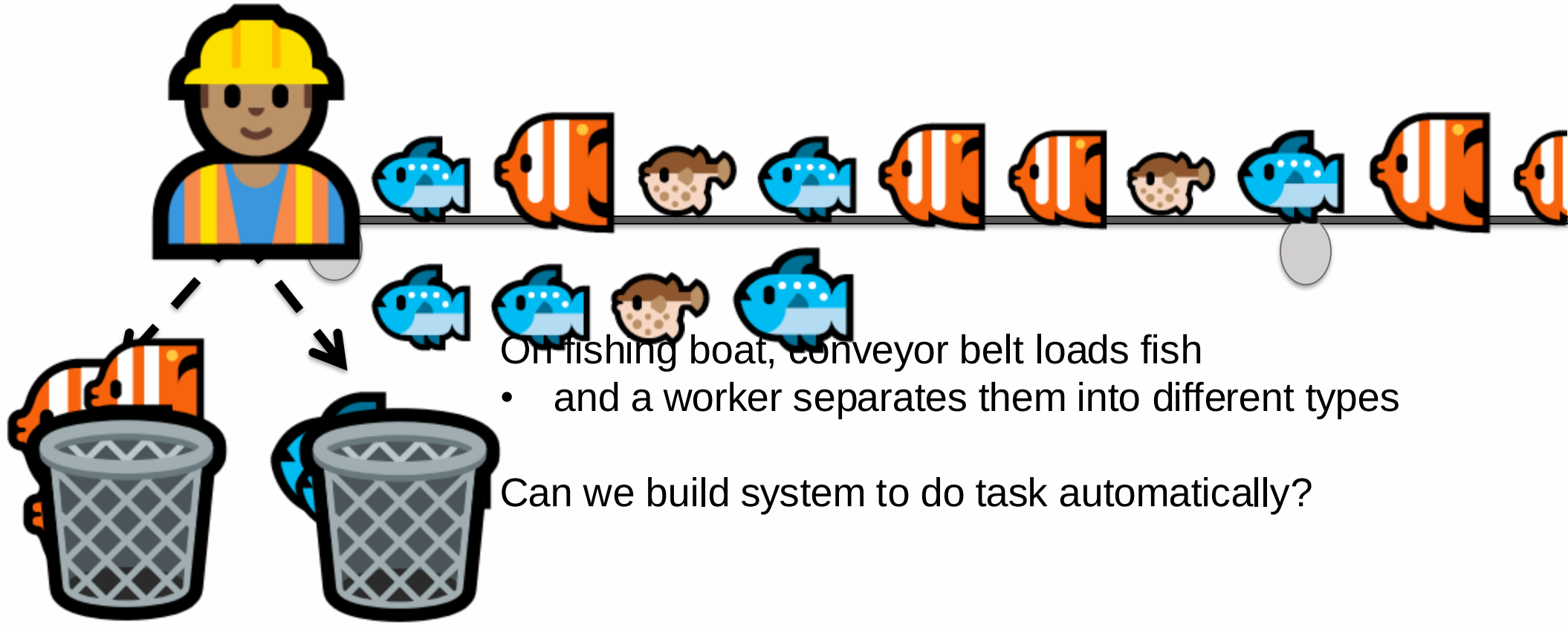
CC reading robot by Mavadee, TH, source: https://thenounproject.com/search/?q=robot%20reading&i=2401182

Tom M. Mitchell 

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

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

Example ML task: categorizing fish

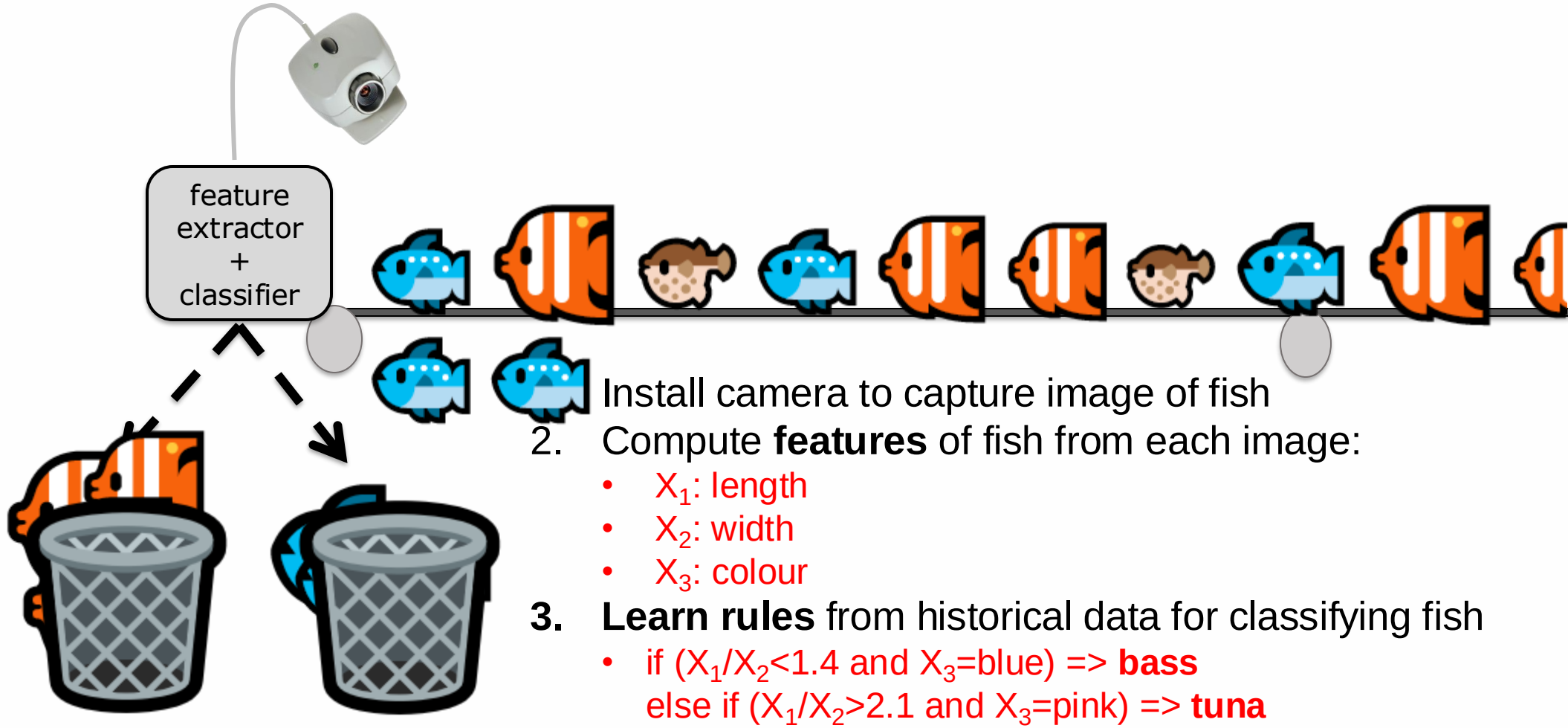


On fishing boat, conveyor belt loads fish

- and a worker separates them into different types

Can we build system to do task automatically?

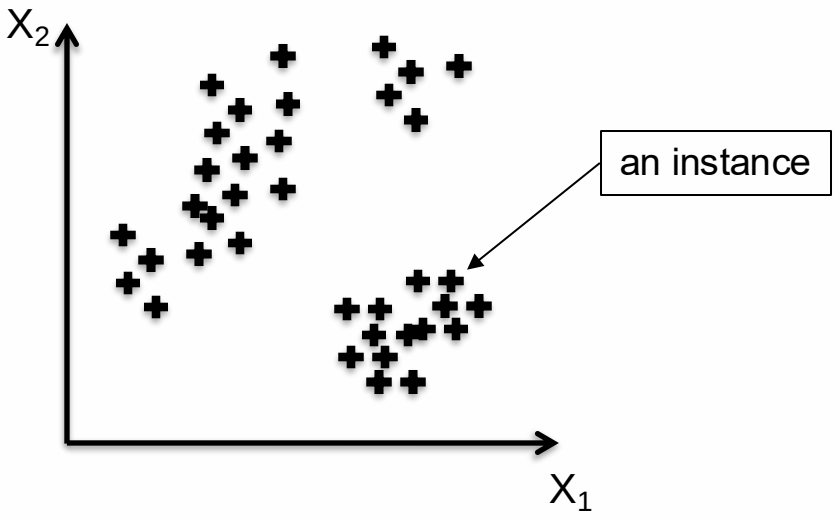
Example: install sensors

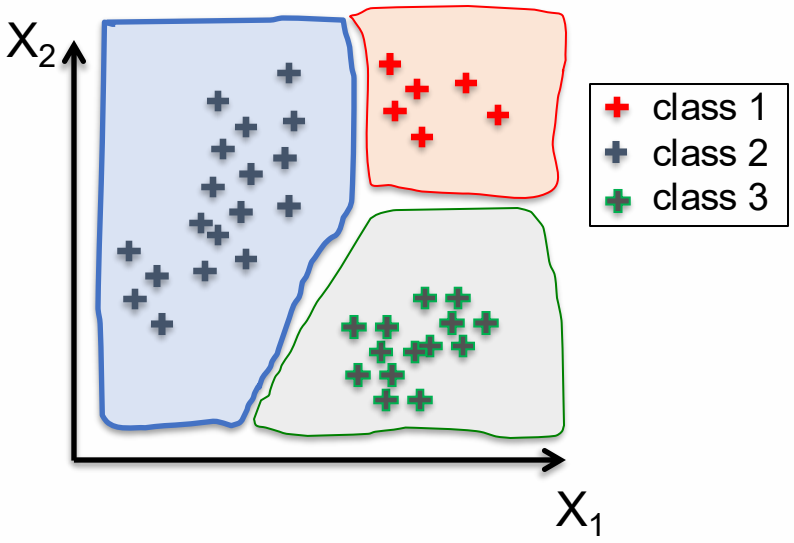


1. Install camera to capture image of fish
2. Compute **features** of fish from each image:
   - X_1: length
   - X_2: width
   - X_3: colour
3. **Learn rules** from historical data for classifying fish
   - if ($X_1/X_2<1.4$ and X_3=blue) => **bass**
     else if ($X_1/X_2>2.1$ and X_3=pink) => **tuna**
     else => **unknown**

Supervised Learning

- Each training instance (fish) is a vector in some **feature space**
- Each training instance has been **labeled** with class (type of fish)
- Task: partition the space for so as to able to make **predictions** for new vectors

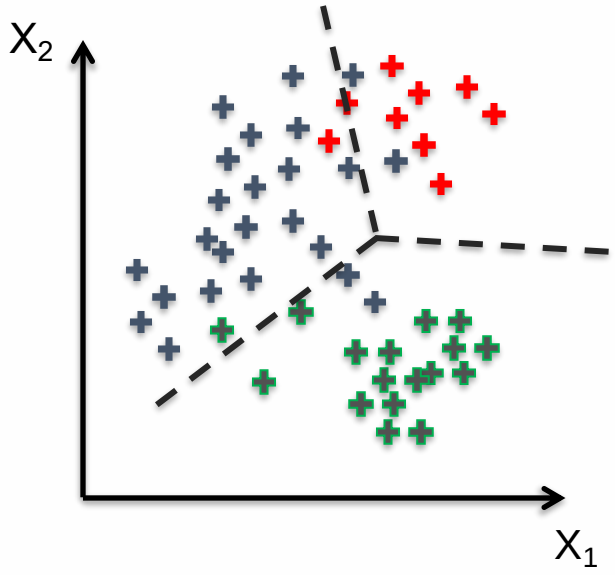




In practice …

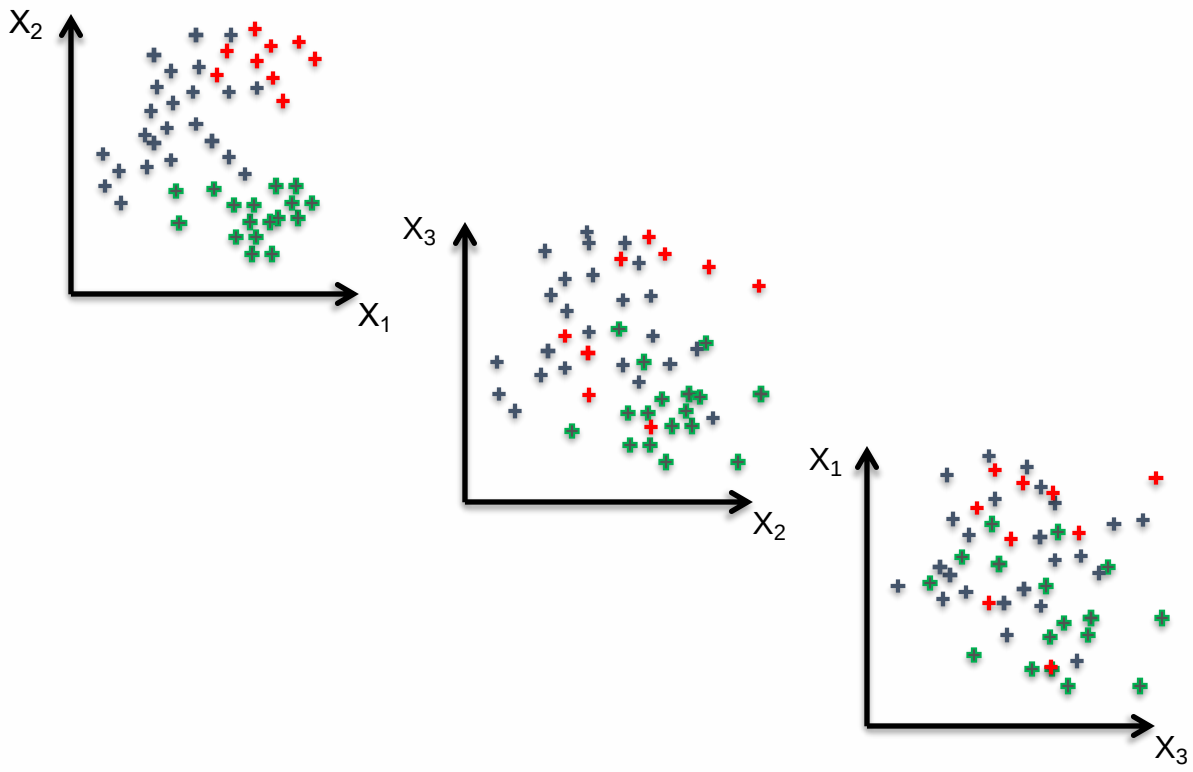
Data is usually overlaps

- so classes are **not linearly separable**



Instances are described by **many features**

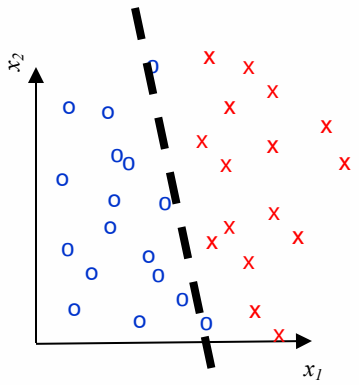
- with some dimensions better at distinguishing classes than others



Types of classification MODELS

All classifiers divide up the feature space

- **boundary** can be **linear** or **non-linear**

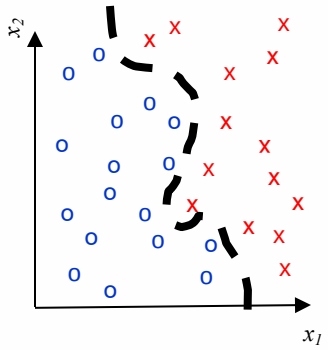


Linear models

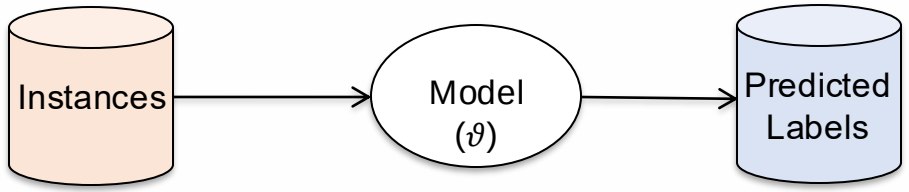
- include Naïve Bayes, Logistic Regression and Support Vector Machine (SVM)

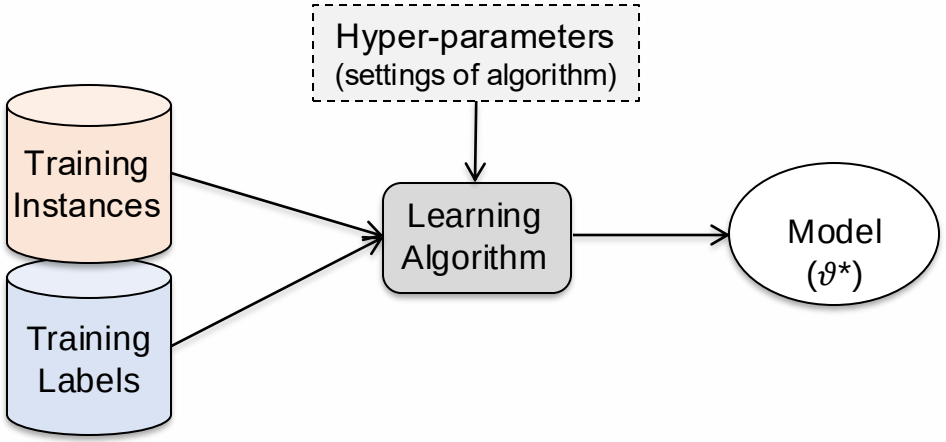
Non-linear models

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



Training a Model





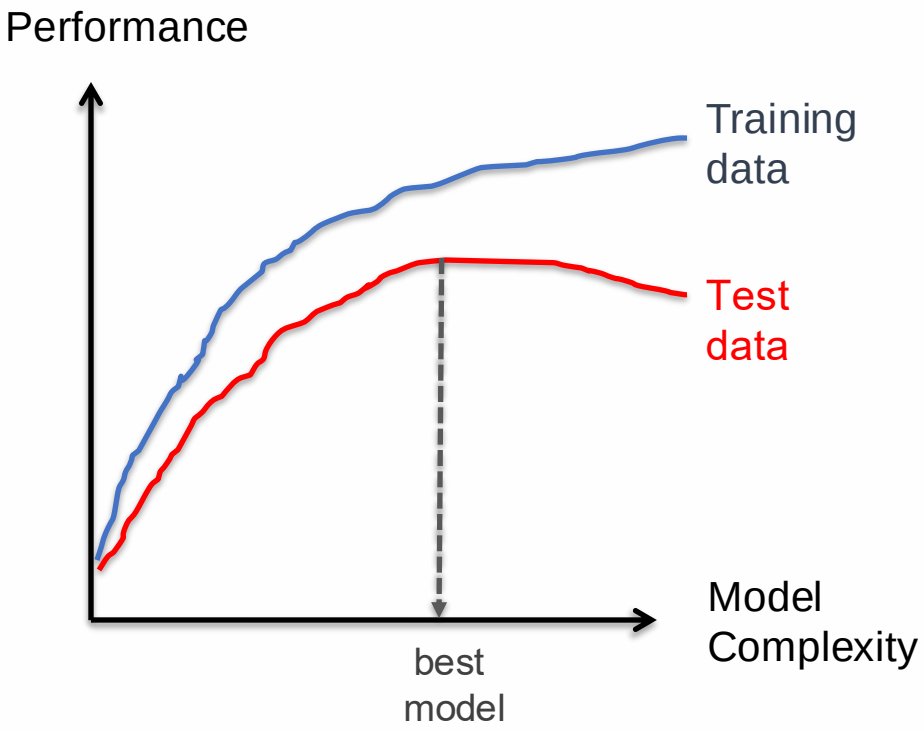
Model

- parameterized formula (e.g. $y = \vartheta^{\mathsf{T}} x$)
- used to predict labels for new instances

Learning algorithm:

- takes as input **training instances** and
- corresponding **ground truth labels**
- searches for **parameters** of model (ϑ^*)
- that minimise **prediction error** (loss) on training labels
- learning algorithm has its own settings (called **hyper-parameters**)

Hyperparameters and Overfitting



Hyperparameters (parameters of learning algorithm)

- can control the **complexity of the model**

Examples of hyper-parameters:

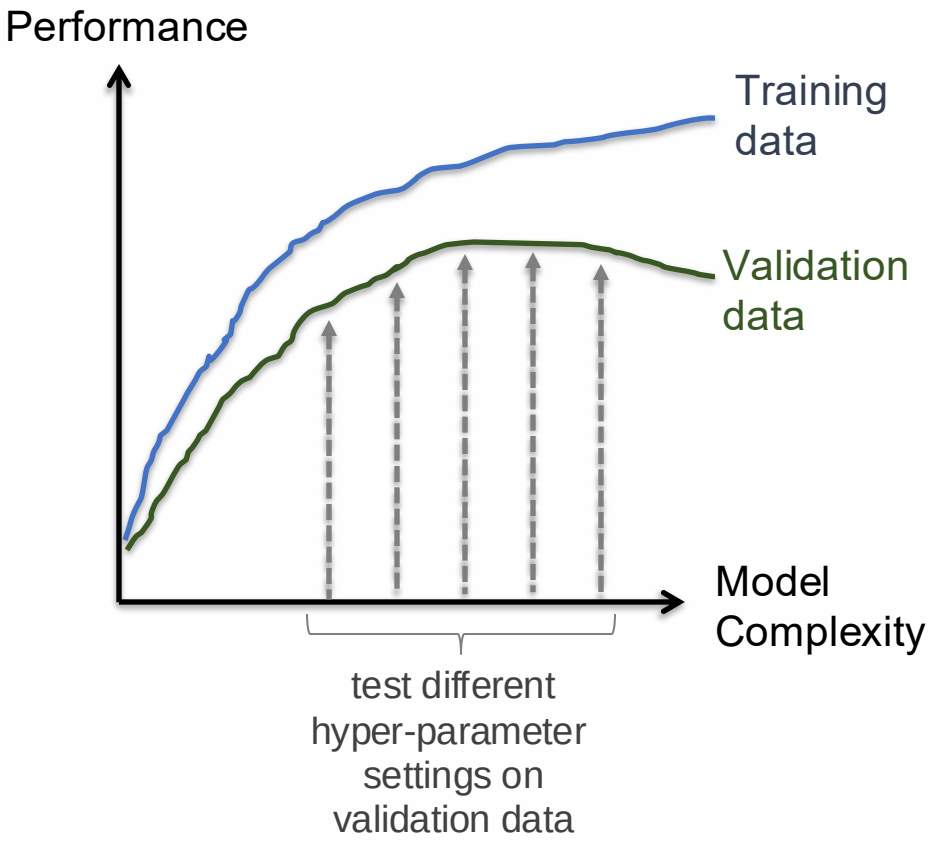
- **#parameters** in model
- **#iterations** over training set and learning rate
- maximum **size** of parameters, etc.

Training error usually improves with **model complexity**

- but **test performance** (error on unseen data) reaches peak
- then degrades as model **overfits** the training set
- by learning spurious patterns from it

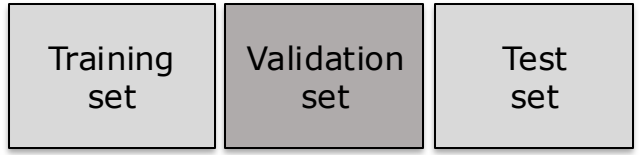
We would like to find the model with the best **generalization performance**

Preventing Overfitting



Choose **hyper-parameter values** to prevent overfitting

- can't use training data, because training error doesn't tell us about generalization
- can't use test data, because we need it later for unbiased testing
- so hold-out portion of training set and use it for interim evaluation:



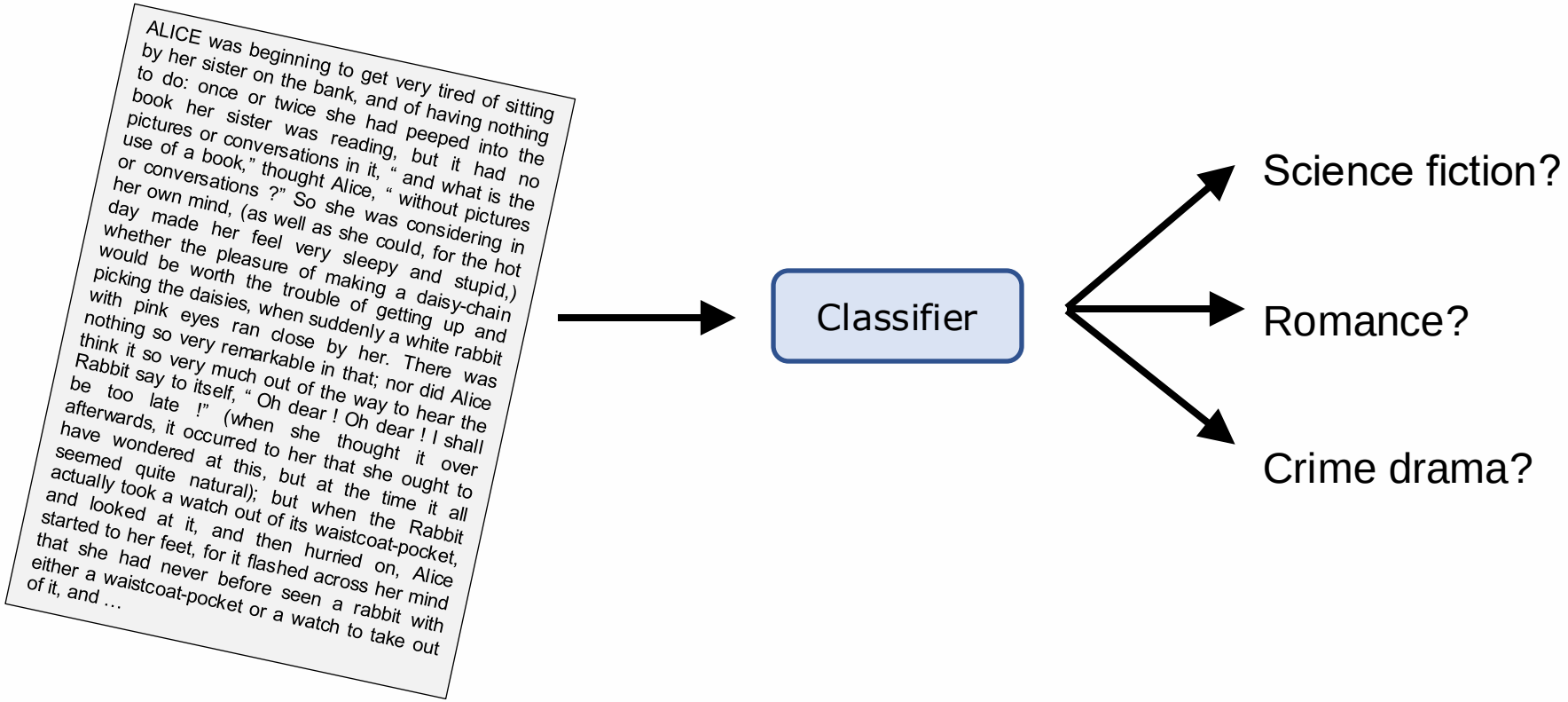
Validation dataset

- used to evaluate different training runs
- choose various hyper-parameter settings, train a model with each, then evaluate on validation set
- keep hyper-parameter values that show best **generalisation 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**

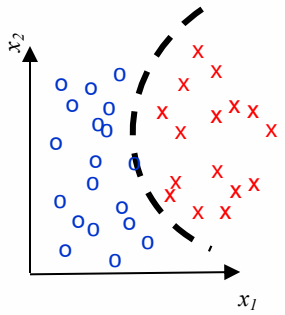
- **spam/phishing detection** – why wouldn't I want to give you my bank account?
- **authorship identification** – who was Satoshi Nakamoto?
- **sentiment analysis** – what didn't the customer like about their stay at the hotel?
- **offensive content detection** – is this post inciting violence?
- **web search query intent** – does the user really want a definition for a library?
- **personalised news feeds** – identify interesting content on social network
- **identifying criminal behaviour online** – could this post be fraud or grooming?
- **routing communication** – find appropriate destination for generic company emails
- **task identification in spoken interfaces** – hey Alexa, can you tell me a joke?
- …

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?

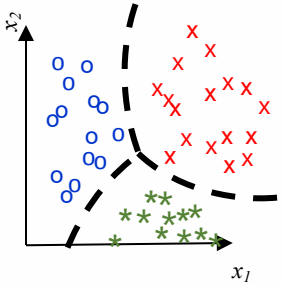
"Hotel room was filthy. Didn't look like it had ever been cleaned" => 1_STAR

Source : http://commons.wikimedia.org/wiki/File:Star_rating_1_of_5.png

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"

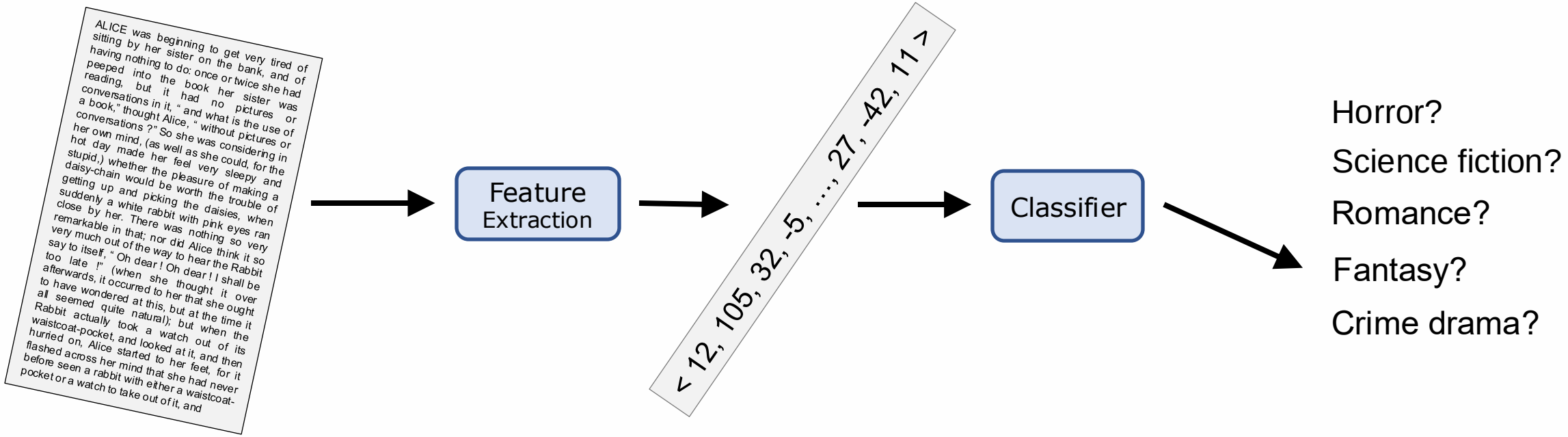
=> POLITICS, SPORT

Extracting Features from Text

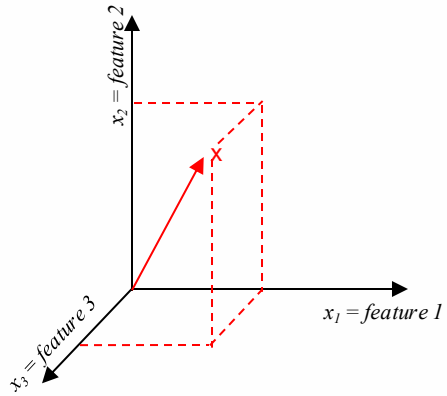
Extracting features from text

Text can be **arbitrarily long**

- no fixed size, means is cannot be given directly to the model
- instead **features** must first be **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 **just the words** themselves

- vocabulary of document provides most 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)

Motivating bag-of-words representations

Consider the following extracts form customer reviews

- which statements do you think are positive and which ones 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 present **in the vocab** of the document

Why not just one-hot encode?

To create a fixed dimension feature vector

- why not just **truncate documents** at fixed length, and treat them as sequence of **categorical variables**?

Once upon a time, in a kingdom where the rivers shimmered like liquid gold and the trees whispered secrets to the wind, there lived a girl named Elara. She was no princess, nor the daughter of a nobleman, but a simple weaver who crafted the most enchanting tapestries in the land.
Legends spoke of her ability to weave not just thread, but the very essence of dreams. Her hands danced over the loom, and with each thread, she captured laughter, sorrow, and even forgotten memories. People traveled from distant lands to see her work, hoping to find a glimpse of their own hearts woven into the fabric….

⇨ *< "Once", "upon", "a", "time", "in", "a", "kingdom", "where", "the", "rivers", …, "tapestries" >*

Encoding categorical variables using **one-hot encoding** produces ***n* binary features** per variable

- where *n* is size of vocabulary
- so sequences of 500 tokens with vocabularies of 50 thousand would produce many millions of columns:

< 1st-word=aardvark?, 1st-word=able?, …, 1st-word=zoom?, 2nd-word=aardvark?, … , 2nd-word=zoom?, … , 500th-word=aardvark?, …, 500th-word=zoom? >

many millions

Too many features given small amount of **training data** available

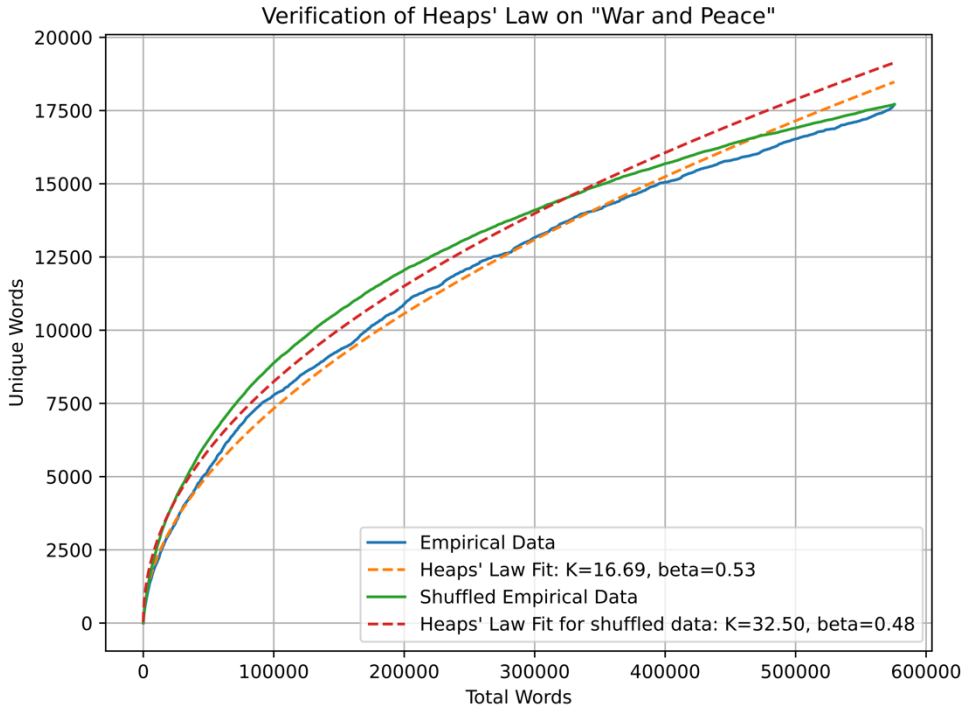
- so to reduce space, **sum** all **one-hot encodings** together
  - reducing #columns to size of vocabulary
  - **throws away** all **word order information** but retains critical vocabulary information

Aside: Word Frequencies

Statistical Laws for Term Frequencies

Heap's law: vocabulary grows with approximately the square root of document/collection length:

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



Zipf's law: token's frequency is approximately proportional to the inverse of its rank:

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

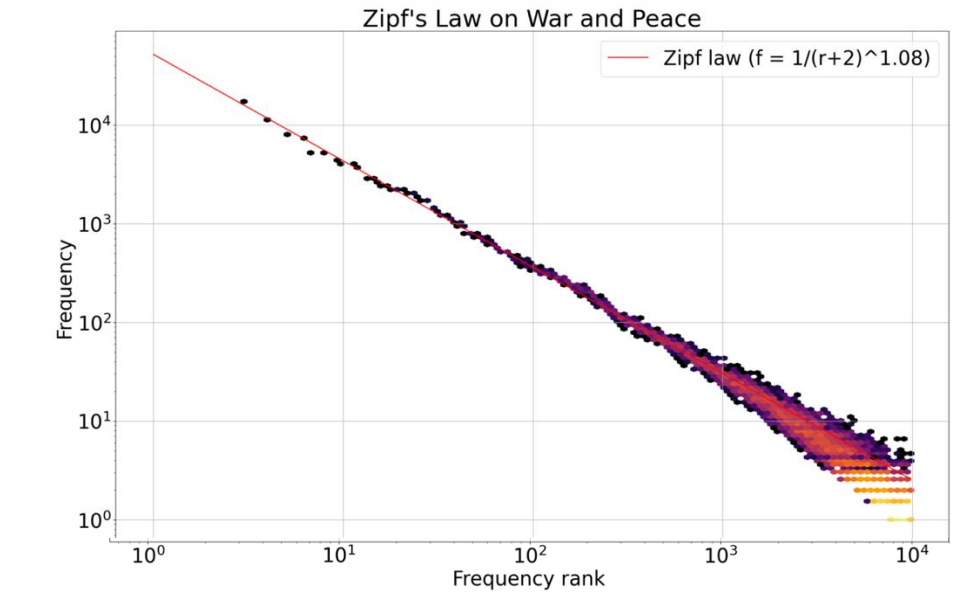


Image sources: https://en.wikipedia.org/wiki/Heaps'_law and: https://en.wikipedia.org/wiki/Zipf%27s_law

Monkeys, Typewriters & Vocabularies

Heap's law derives from Zipf's law and can be explained by **random typing model**

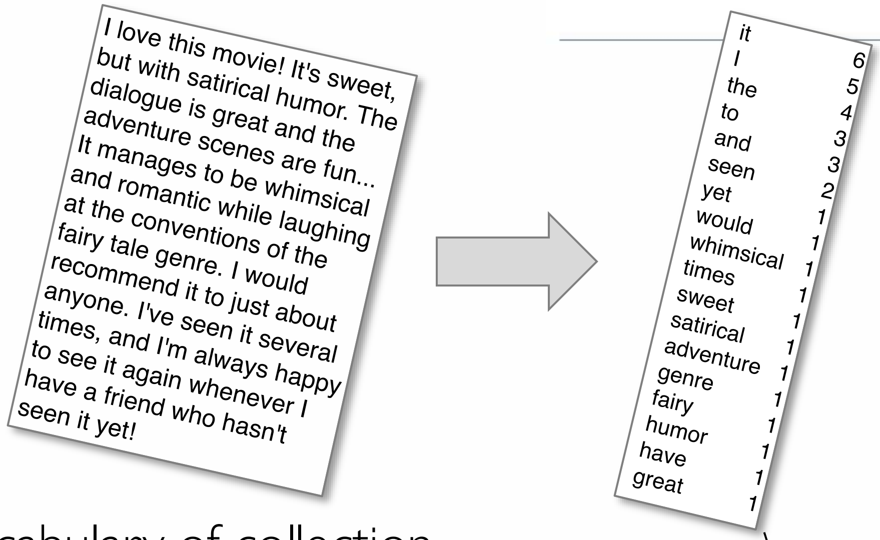
- a.k.a. Monkeys at typewriters (https://www.cs.cmu.edu/~zollmann/publications/monkeypaper.pdf)
- and if you wait long enough and have sufficient number of monkeys, eventually one will produce Shakespeare …
  - Infinite monkey theorem (https://en.wikipedia.org/wiki/Infinite_monkey_theorem)

Point is, vocabulary of document/collection **grows slowly** compared to its length

- examples of collections and corresponding vocabulary sizes:

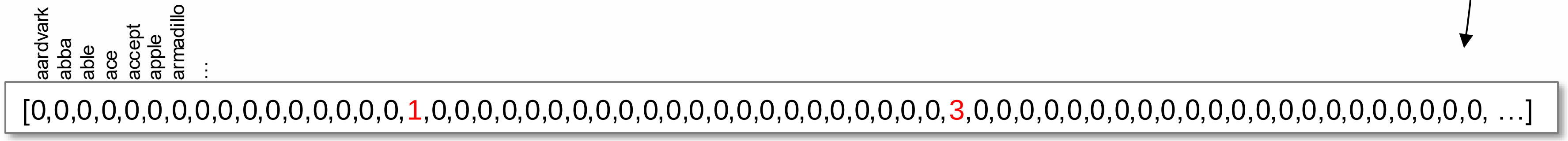
	Tokens	Vocabulary
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884 thousand	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

Bag-of-words (BOW) representation



Document vocabulary is **very small** compared to vocabulary of collection

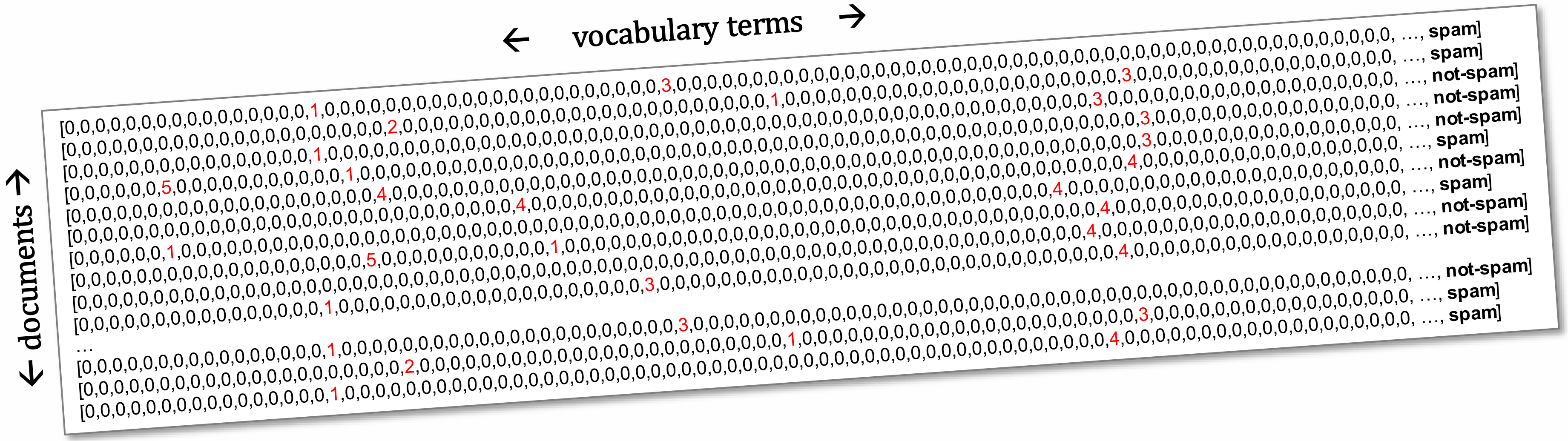
- so terms present in document usually **characterise** well content of document
- in BOW representation include also count of occurrences of each term
  - could use binary representation of document with little information loss
- very sparse representation:



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**

Tokenizing & Preprocessing Text for Building Classifiers

Simple Tokenizers in Python

Default tokenizer in scikit-learn toolkit for ML in Python:

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

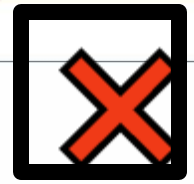
Common Pre-processing of Text

Common pre-processing activities:

- Prior to tokenisation:
  - Remove **mark-up** (non-content information), e.g. <HTML> tags
  - **Lowercase** the text to reduce vocabulary size (can lose information: e.g. 'WHO' vs 'who')
  - Remove **punctuation** (;,&%$@!><? etc.)
- After tokenisation:
  - Remove **stopwords** (extremely high frequency words)
  - Remove very **low frequency words** (since insufficient information present to determine correlation between word presence and class)
- Less common activities when building classifiers:
  - Perform **stemming** or **lemmatization** to reduce vocabulary size
  - Perform **spelling correction**

Aside: Spelling Correction

Probabilistic Spelling Correction



Consider the phrase: *... stellar and versatile* acress *whose combination ...*

- observed word *"acress"* is **not** in **vocabulary,** what is chance writer intended *"**actress**"*?
- if we had enormous corpus of misspellings and corrections, could estimate **relative frequency**:

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

- but we don't have such a corpus …

Could estimate probability using string **edit distance**

- by counting number of insertions, deletions, substitutions or transpositions needed to get from one string to the other
- but then **actress, caress, cress, access, across, acres** would all be equally likely
- surely some words (like **actress**) are more likely than others (like **cress**) …

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

Spelling Correction: Bayes Rule

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 prior, *P(correct)*
  - 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

Smoothed estimate of prior probability

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

- the likelihood, *P(observed|correct)*
  - estimate $P(observed=\text{“}acress\text{”}|correct=\text{“}across\text{”})$ by counting in large corpus of errors how many times *e* has been substituted for *o*
  - need also confusion matrix for deletions, insertions and transpositions

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	0	0	7	0	6	3	1	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

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 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 context leads to NB Classifier

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

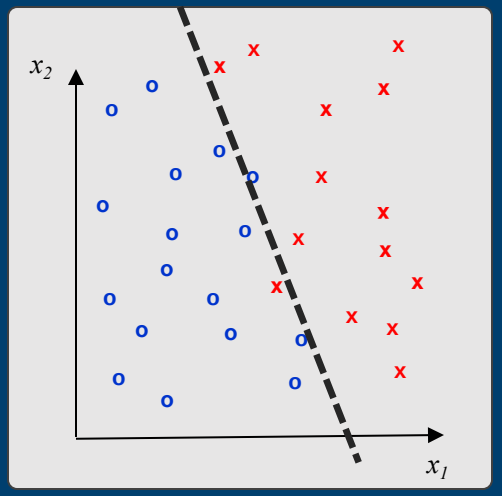
... was called a "stellar and versatile acress whose combination of sass and glamor has defined her ..."

How can we make use of that context information?

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

    P(bigram="versatile actress") = P(previous="versatile", correct="actress")
    = P(previous="versatile" | correct="actress") P(correct="actress")
  - so we have:

    P(correct| observed, previous) ∝
    P(observed="acress" | correct="actress") P(previous="versatile" | correct="actress") P(correct="actress")
- which is just a **Naïve Bayes model** with two features
  - the observed (incorrect) word and the previous word in the sentence



Linear Classification Models

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

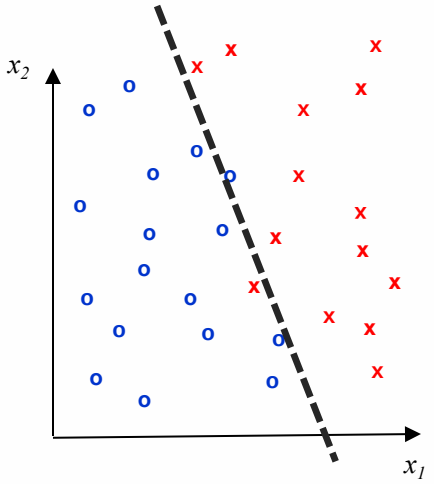
Linear Classification Algorithms

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

stamp corpulent popularity by geralt (Pixabay Licence)

Linear models estimate one parameter per vocabulary word

- making them **highly interpretable**
- can see which vocabulary terms influence prediction and by how much

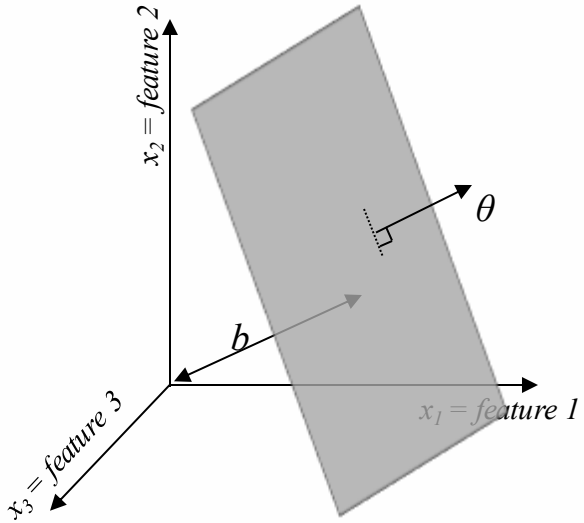


Now discuss quickly the 3 most popular **linear models**:

- **Naive Bayes** – traditionally used with text
- **Logistic Regression** – works well if regularized
- **Support Vector Machines** – naturally regularized and designed for high dimensional data

Decision Boundaries: hyperplanes

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



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

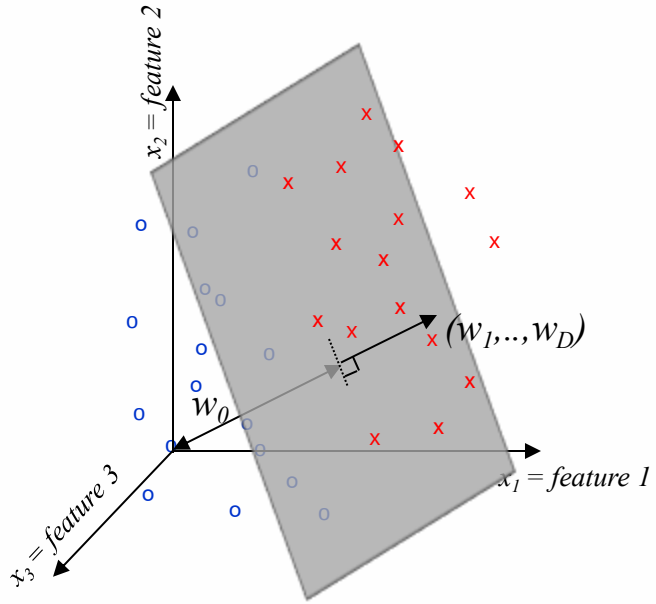
- and a set of ***model parameters*** denoted θ,
- the oriented hyperplane has equation:

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

- 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 $b = -\theta_0$

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(*spam | *hi mark I am a student in your nlp course and …)*

- can't estimate class probability directly, because haven't seen any email **exactly** like it before, so use Bayes' rule to swap the order of the events:

  = $\frac{P(\textit{hi mark I am a student in your nlp course and …}|\text{spam})P(\text{spam})}{P(\textit{"hi mark I am a student in your nlp course and …"})}$

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

  ∝ *P(hi mark I am a student in your nlp course and …|*spam*)P(*spam*)*

- make **simplifying** (and clearly wrong) **assumption** that words occurrences are not correlated within the classes
  - i.e. assume *P(hi | mark I am a student in your nlp course and… ,* spam*) = P(hi |* spam*)*

    ∝ *P(hi |* spam*) P(mark |* spam*) P(I |* spam*) P(am |* spam*) P(a |* spam*) … P(*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 *a* for each feature as follows:

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

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

Is independence assumption a big deal? No …

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 perform quite as well on large data as other classifiers
  - since redundant features being counted twice!
- predicted probabilities are not well calibrated
  - predictions are often **overconfident** due to violations of independence assumption

Linear Classifiers: Logistic Regression

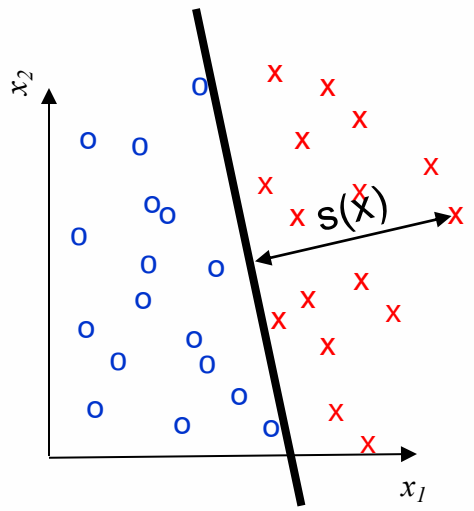
Distance from Hyperplane

The further a point is from decision boundary

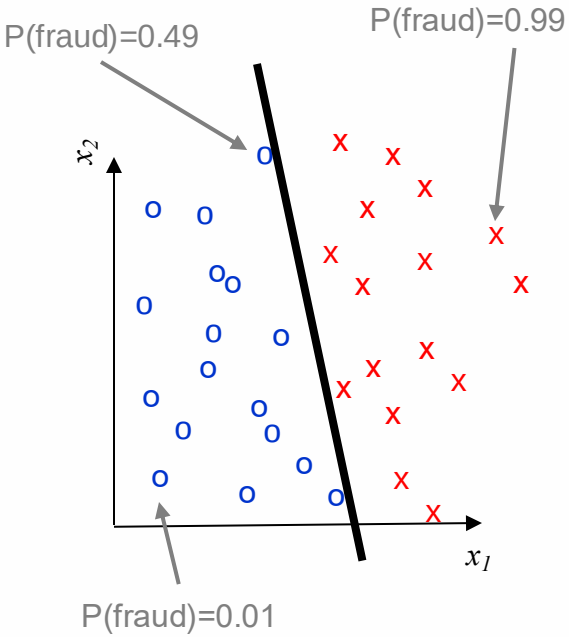
- the more certain we are about the prediction

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



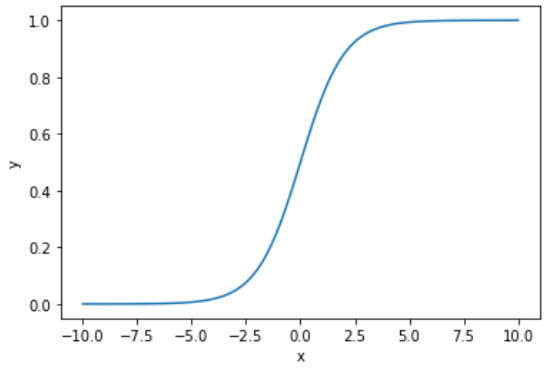
Instead of mapping each point to *fraud/non-fraud*

- 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

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:

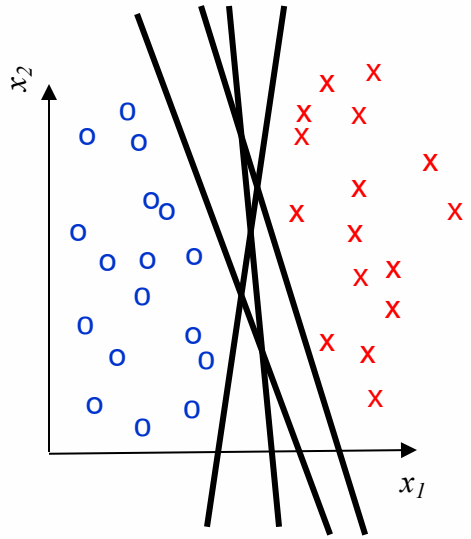
- assumes feature values are ***linearly related*** to *log odds*
  - if assumption is *strongly* violated, model will perform poorly

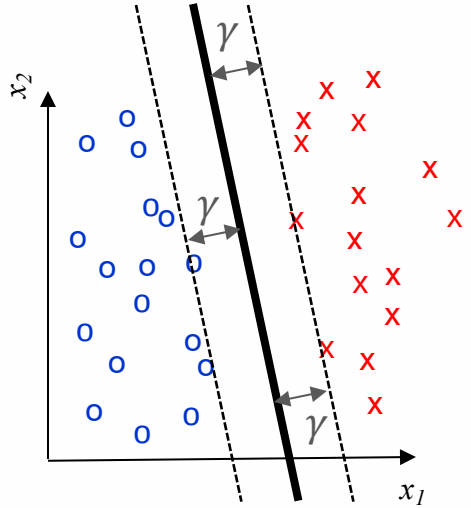
Linear Classifiers:
Support Vector Machines

Maximising the margin

Imagine data for 2 classes splits nicely into 2 groups:

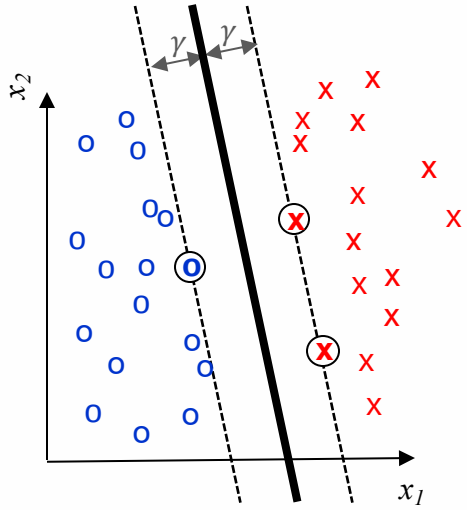
- lots of options for where to place linear decision boundary
  - which one should we choose?





- 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



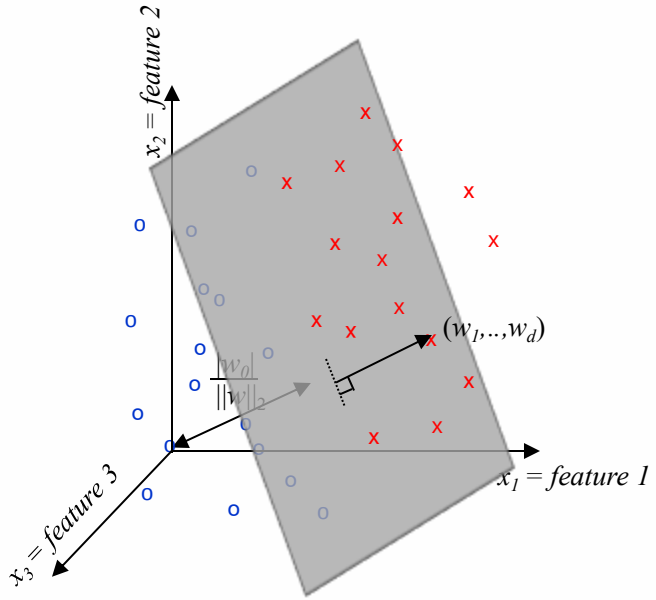
Points lying exactly on margin referred to as **support vectors**

- 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)



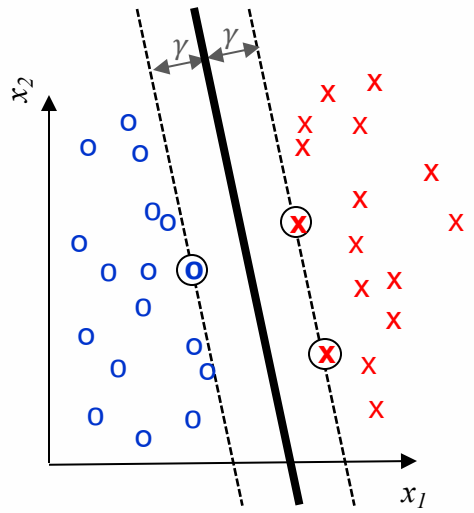
So basic *support vector machine* (SVM) is also linear classifier

- 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

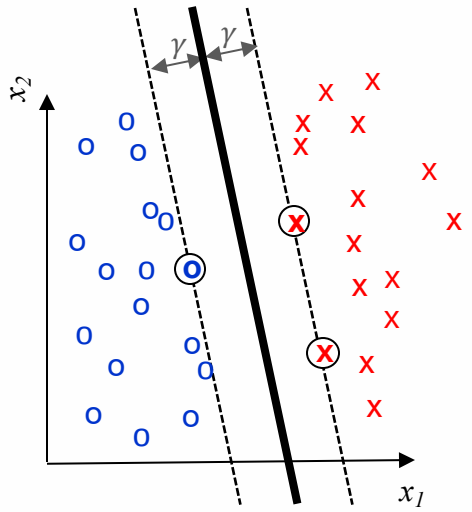


We can compute the distance to points from the hyperplane using:

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

- 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: $\forall_j\ y_j\boldsymbol{w}\cdot\boldsymbol{x}_j \geq 1$

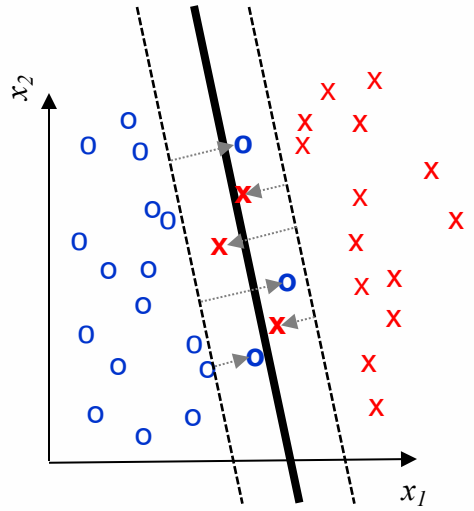
Mathematically, same as minimizing loss: $\sum_i w_i^2 + \sum_j \varepsilon_j$

- 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**?

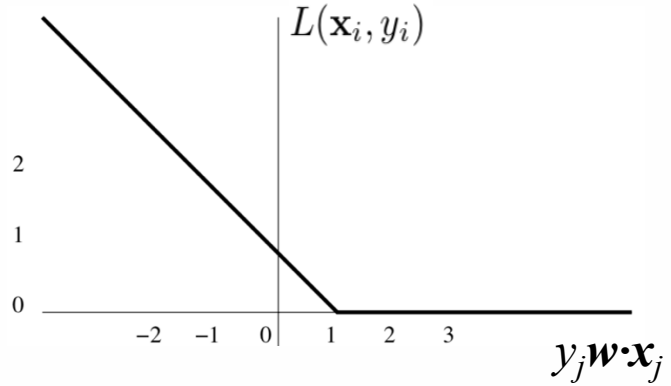
- simply penalise points on wrong side of **margin** based on distance from it
- *support vectors:* points on wrong side of margin that provide non-zero contribution to loss function
- objective function to minimise remains almost unchanged:



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

- ϵ_i is distance from i^{th} support vector to margin
- C is a 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:

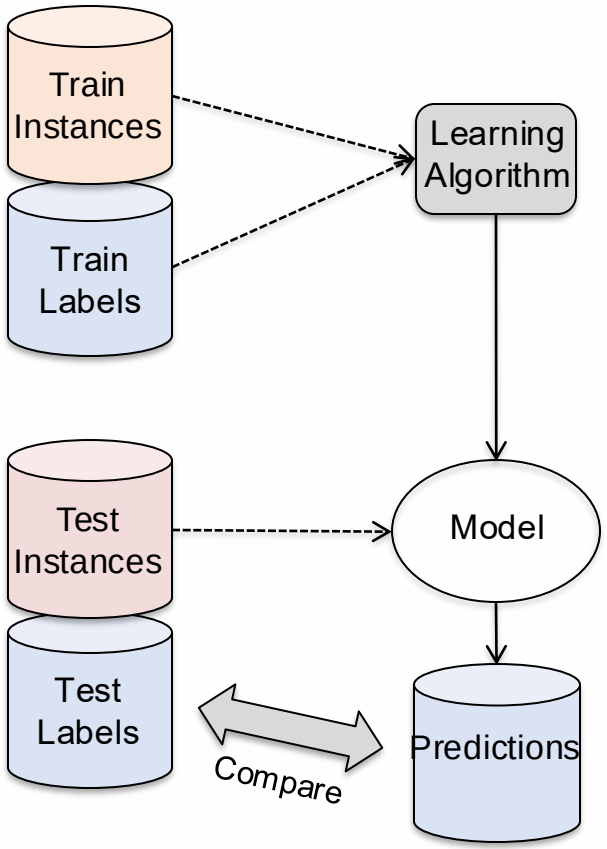
- Naive Bayes
- Logistic Regression
- Support Vector Machines

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

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

Evaluating Text Classifiers

Training and Testing

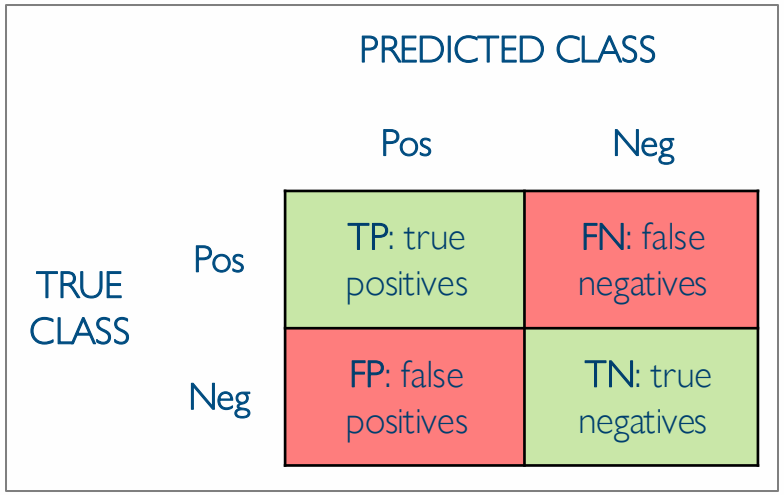


Test model:

- by comparing predicted labels with ground truth labels for instances from the test set

Evaluating a Binary Text Classifier

First thing to investigate is 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
- **AuC**
  = Area under the ROC Curve is often used as a single measure that doesn't depend on the confidence threshold used to make predictions with the model

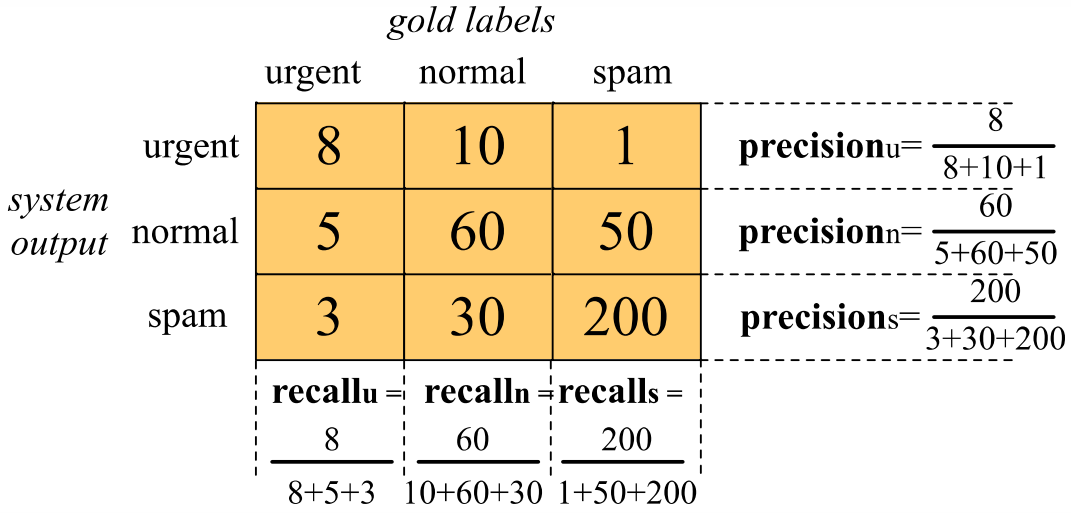
$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

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

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

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

Evaluating Multi-class Classifiers



If there are *n* classes, confusion matrix will be *nxn*

- calculate a precision/recall value for each class
- by considering it positive class in a one-versus-all setting

Combine each class's Precision and Recall values into a single measure:

- **Macro-average**: average over classes weighting each class the same
- **Micro-average**: average over classes weighting each class by the number of datapoints in it

Conclusions on Text Classification

Conclusions on Text Classification

Traditionally use a **bag-of-words representation** of text documents

- And linear classifiers such as a Support Vector Machine or regularized Logistic Regression