ECS763P
Natural Language Processing

Week 2
Text Classification

Matthew Purver
(with material from Jurafsky & Martin book, Ng machine learning course)

Coursework 1

- Lab sessions weeks 2-5:

- Train & test an SVM-based anger detector
- Build a simple language model
- Build a simple topic model
- Apply them to see what people were angry about during last week's tube strike

- See you on Wednesday!

Text Classification Tasks

- Classification is the core method in:
  - Spam filtering
  - Document categorisation
  - Language identification
  - Sentiment analysis
- And a key component in:
  - Dialogue systems
  - Opinion mining
  - Author identification
  - …

Text Classification: definition

- *Input*:
  - a document *d*
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
- *Output*: a predicted class $c \in C$
- *Classifier:* a function $\gamma: d \rightarrow c$
- *Example*: spam filtering
  - $C = \{spam, not\_spam\}$

Rule-based classifiers

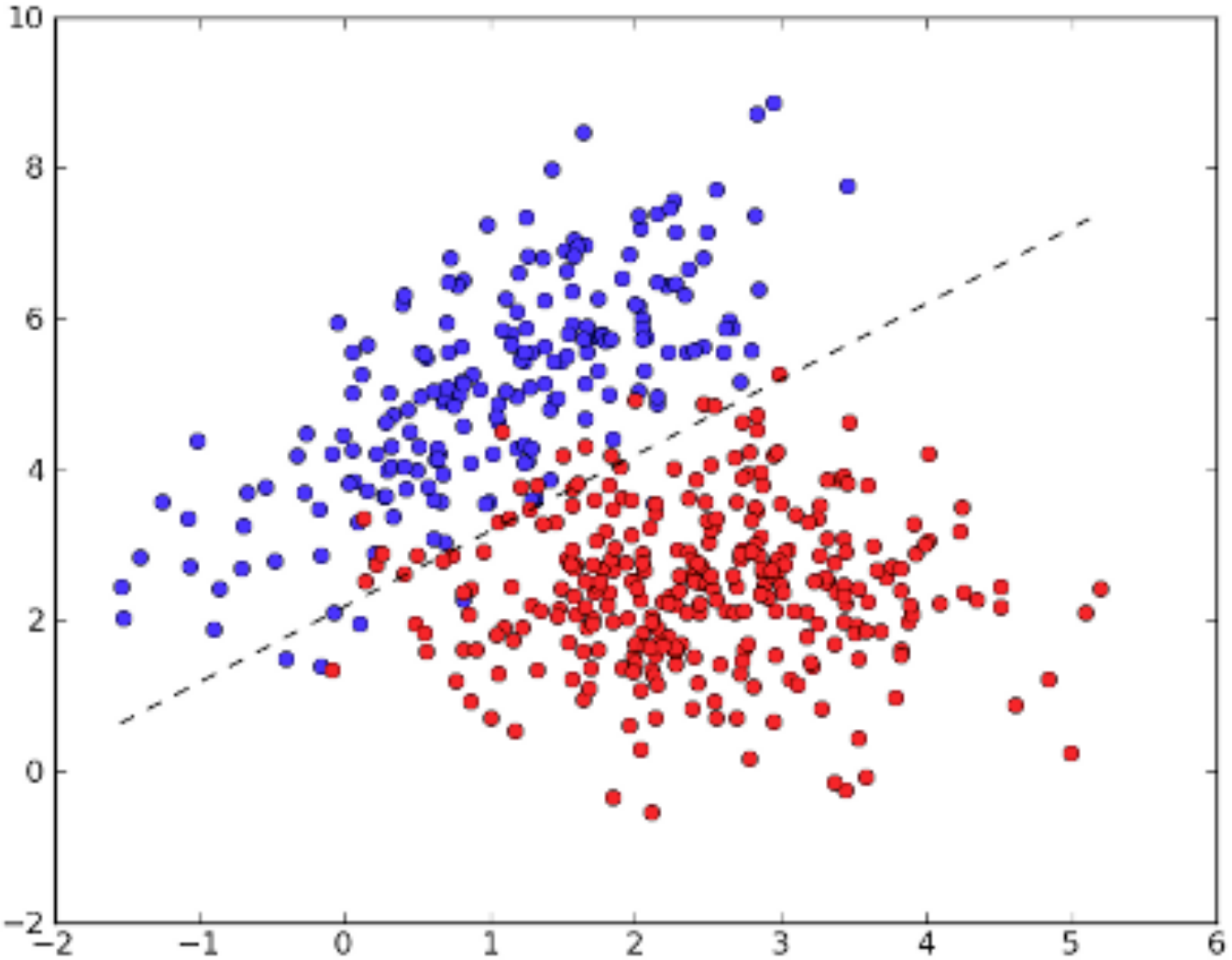
- We could use a set of rules:
  - Simple dictionary lookup
  - More complex (weighted) combinations
- Manual
  - Can be high accuracy
  - Requires expert knowledge and high maintenance
- Automatically derived
  - E.g. rule-based learners, decision trees
    - (see Data Mining lectures)
  - Usually brittle unless in restricted domains

Statistical classifiers

- Supervised machine learning
- *Input*:
  - a document *d*
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
  - a training set of *m* hand-labeled documents $(d_1,c_1),\ldots,(d_m,c_m)$

- *Output*: a learned classifier $\gamma: d \rightarrow c$

Linear classifiers



mlpy.sourceforge.net

Bayes' Rule

$$P(c,d) = P(c \mid d)P(d) = P(d \mid c)P(c)$$

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)}$$

- Generative models:

Naïve Bayes

- Bayes' Rule:
  - For a document *d* and a class *c*

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)}$$

- MAP (maximum a-posteriori) classifier:

$$c_{MAP} = \operatorname*{argmax}_{c \in C} P(c \mid d) = \operatorname*{argmax}_{c \in C} \frac{P(d \mid c)P(c)}{P(d)}$$

$$= \operatorname*{argmax}_{c \in C} P(d \mid c)P(c)$$

- Document as features x_1, x_n

$$= \operatorname*{argmax}_{c \in C} P(x_1, x_2, \ldots, x_n \mid c)P(c)$$

Naïve Bayes

- But how can we calculate $P(x_1, x_2, \ldots, x_n \mid c)$
  - (how often will we see it?)
- Instead, assume:
  - features are unordered words ("bag of words")
  - features are conditionally independent

$$P(x_1, \ldots, x_n \mid c) = P(x_1 \mid c) \bullet P(x_2 \mid c) \bullet P(x_3 \mid c) \bullet \ldots \bullet P(x_n \mid c)$$

- So:

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_{i \in positions} P(x_i \mid c_j)$$

$$= \underset{c_j \in C}{\operatorname{argmax}} \log(P(c_j)) + \sum_{i \in positions} \log(P(x_i \mid c_j))$$

Learning a Naïve Bayes Model

- First attempt: maximum likelihood estimates
  - simply use the frequencies in the data

$$\hat{P}(c_j) = \frac{doccount(C = c_j)}{N_{doc}} \qquad \hat{P}(w_i \mid c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)}$$

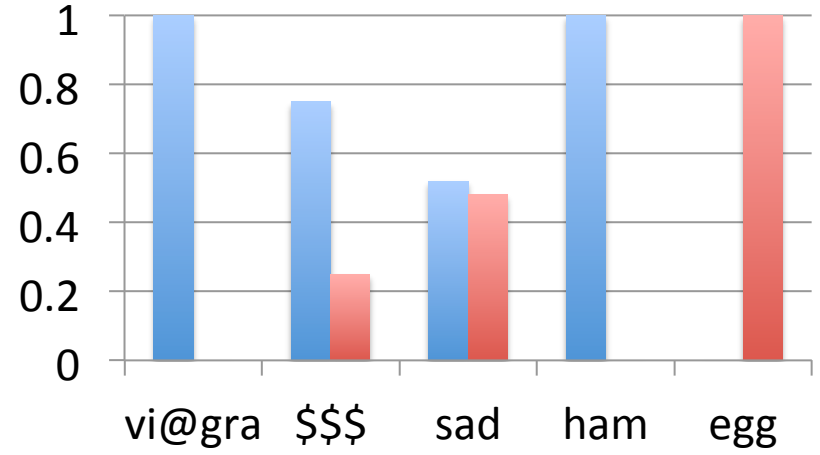
  - "maximum likelihood model"

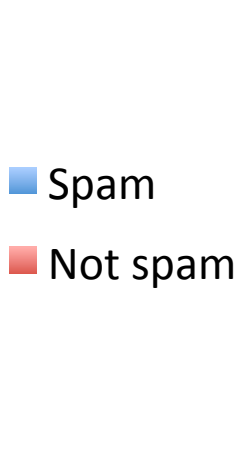
- But what if we haven't seen "viagra" with c = not_spam?

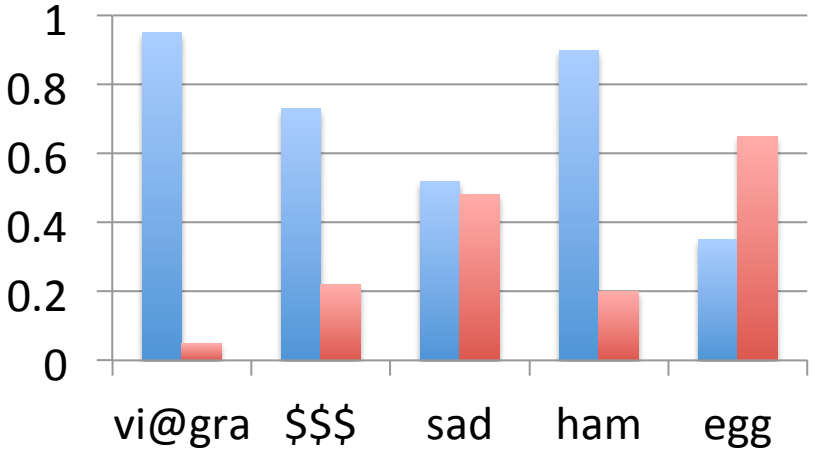
$$\hat{P}(w_i \mid c_j) = 0!$$

Smoothing

- Need to reserve some probability mass for unobserved events







- E.g. Laplace (add-1, add-N) smoothing for Naïve Bayes

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} \left(count(w, c) + 1 \right)} = \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c) \right) + |V|}$$

Worked example

	#DOCS	viagra	$$$	help	matt	hi	and	email	nlp
SPAM	50	24	30	14	7	33	50	23	0
NOT	200	1	0	21	150	129	200	51	32

	P(c)	P(w|c)							
SPAM	0.2								
NOT	0.8								

Worked example

	#DOCS	viagra	$$$	help	matt	hi	and	email	nlp
SPAM	50	24	30	14	7	33	50	23	0
NOT	200	1	0	21	150	129	200	51	32

	P(c)	P(w|c)							
SPAM	0.2	0.48	0.60	0.28	0.14	0.60	1.0	0.46	0.00
NOT	0.8	0.01	0.00	0.11	0.75	0.65	1.0	0.26	0.16

Worked example

	#DOCS	viagra	$$$	help	matt	hi	and	email	nlp
SPAM	50	24	30	14	7	33	50	23	0
NOT	200	1	0	21	150	129	200	51	32

	P(c)	P(w|c)							
SPAM	0.2	0.48	0.60	0.28	0.14	0.60	1.0	0.46	0.00
NOT	0.8	0.01	0.00	0.11	0.75	0.65	1.0	0.26	0.16

	log	log							
SPAM	-1.61	-0.74	-0.51	-1.27	-1.97	-0.51	0.00	-0.78	-Inf
NOT	-0.22	-5.30	-Inf	-2.25	-0.29	-0.44	0.00	-1.37	-1.35

- Can we classify:

  `hi matt`

  `hi matt want some viagra?`

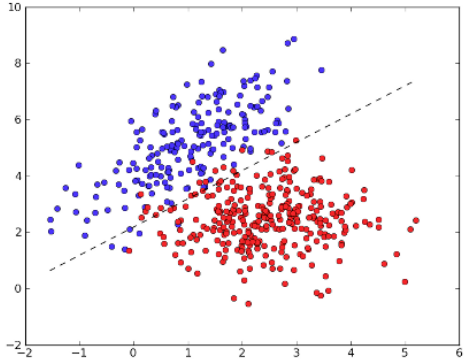
Naïve Bayes: Summary

- Very fast, low storage requirements
- A good dependable baseline for text classification
- Robust to irrelevant features
  - Irrelevant features cancel each other without affecting results
- Very good in domains with many equally important features
  - Decision Trees suffer from fragmentation in such cases – especially if little data
- But often the independence assumptions **don't hold**
  - Other classifiers usually give better accuracy

Logistic Regression

- Naïve Bayes is a linear classifier
  - Decision is a linear function of weighted feature values
- But:
  - we calculate feature weights & boundary independently
    - (i.e. we're assuming statistical independence)
  - we're using a generative model

$$y = \text{argmax}_y P(y|x) = \text{argmax}_y P(x|y).P(y)$$



- Why not:
  - estimate the boundary directly?
  - use a discriminative model? $y = \text{argmax}_y P(y|x)$

Logistic Regression

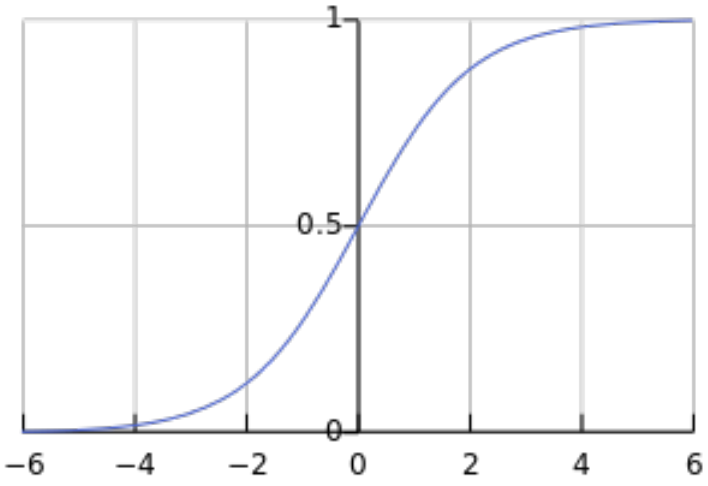
- Output is a linear combination of features
  - i.e. weight matrix w, feature vector x

$$p(y \mid x) = \sum_i w_i . x_i$$

$$p(y \mid x) = \vec{w} . \vec{x}$$

- … with exponential to make it positive …
- … and normalised to [0-1]

$$p(y \mid x) = \frac{1}{1 + e^{-\vec{w} . \vec{x}}}$$



- Now learn the optimal w

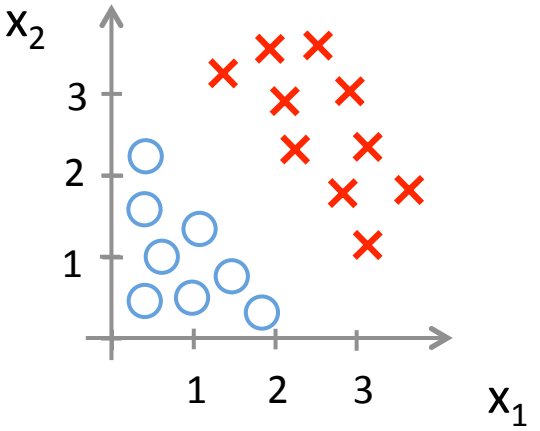
Logistic Regression

- Output $$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

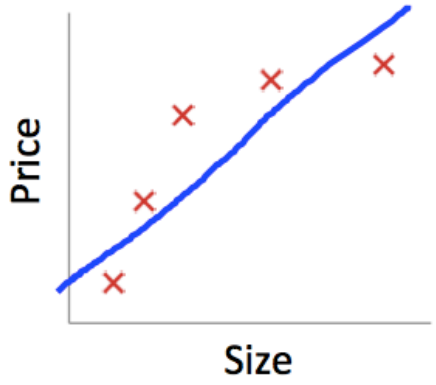
- Error function $$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

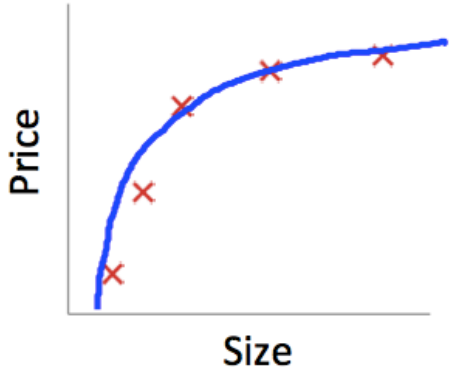
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

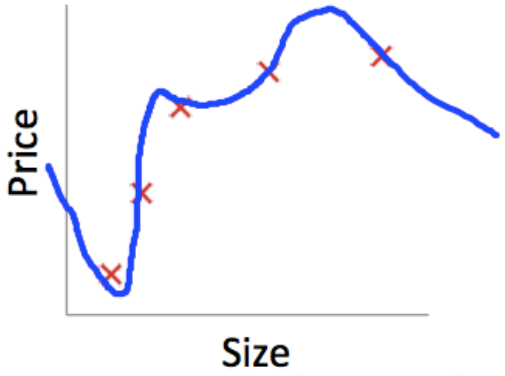
- We want $\min_\theta J(\theta)$
- We can find it by gradient descent
  - given a labelled training set



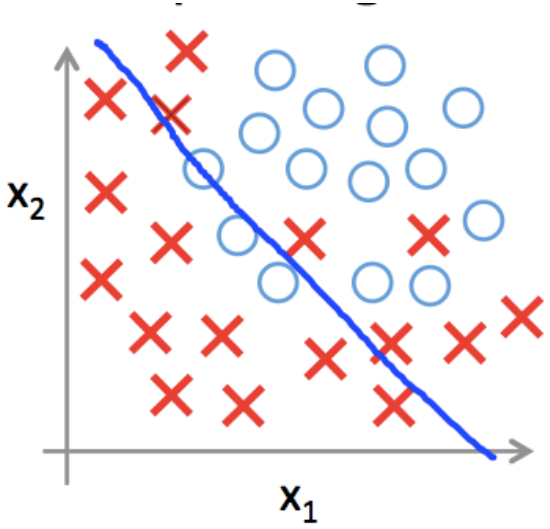
Overfitting

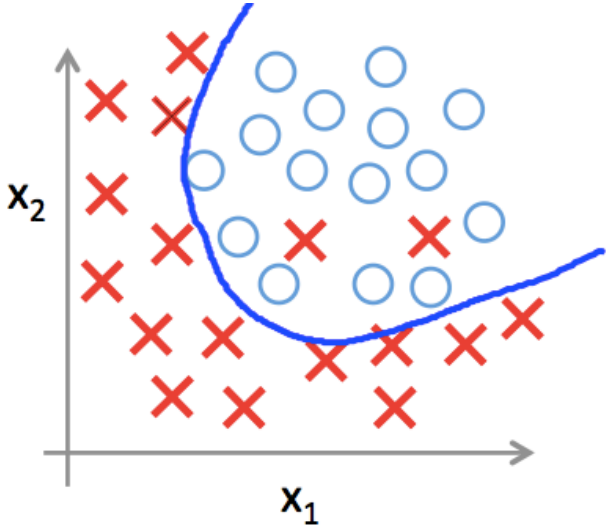


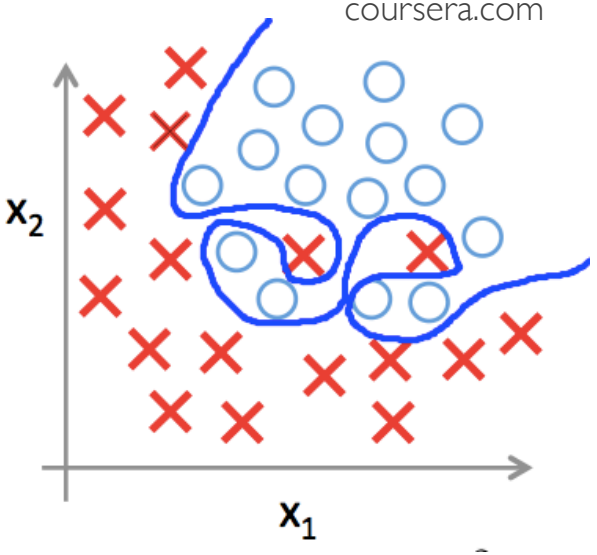




Andrew Ng, coursera.com







Regularisation

- Logistic regression cost function:

$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)} + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))\right]$$

- Regularisation: add in a term for weights learned:

$$J(\theta) = \left[-\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log (h_\theta(x^{(i)}) + (1 - y^{(i)}) \log 1 - h_\theta(x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

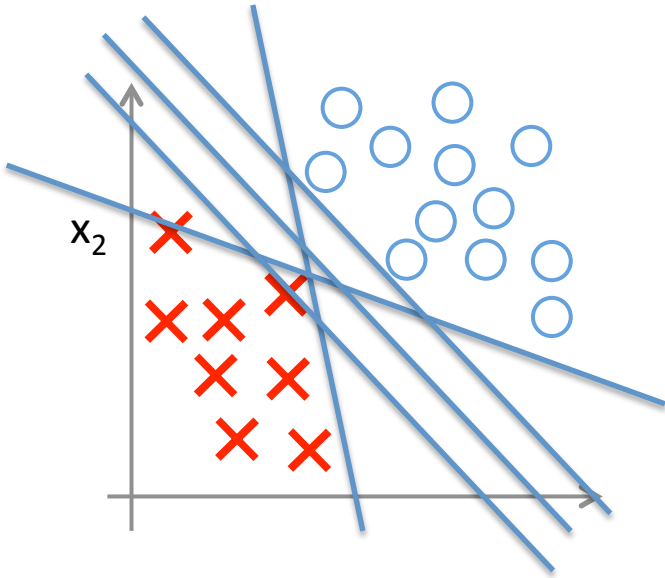
- Smaller parameter values

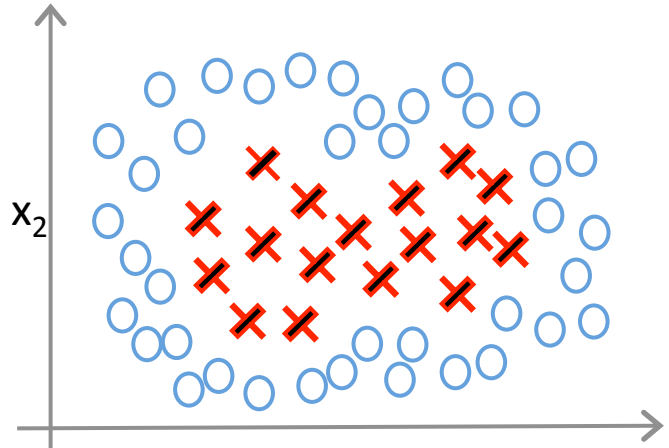
  = “simpler” hypothesis

  = less prone to overfitting

Support Vector Machines

- Like logistic regression, but:
  - Maximum margin criterion
    - (via different form of cost function)
    - choice of parameter C
  - Kernel function
    - (allows easy calculation of non-linear functions)
    - choice of kernel
      - Linear
      - Polynomial
      - Radial basis function





Classifiers: Summary

- Naïve Bayes:
  - Not particularly high accuracy (false independence assumption)
  - Fast, easy, dependable baseline
  - Be careful with smoothing!
- Logistic Regression:
  - Higher accuracy
  - Good for small numbers of features
  - Be careful with regularisation!
- Support Vector Machines:
  - Good for large numbers of features
  - Very robust, dependable, good accuracy
  - Be careful with choice of cost parameter C, kernel function

Features

- All classifiers can only be as good as their features
  - What do we use?

- Bag of words assumption
  - Good baseline!
- N-gram features
  - Subsequences of N consecutive words
  - Often with START, END symbols
- Weighting?
  - Binary features: 1 if present, 0 if not
  - Count-based features: frequency, log(frequency)
  - Text length normalisation
  - TF-IDF weighting
    - Term frequency = how often does term appear in this text?
    - Inverse document frequency = how rare are texts containing term?

Tokenisation & Normalisation

- We want to split a string into words
  - Tokenisation
    - `I thought “I like it!”`
      → `[I, thought, I, like, it]`
    - First try: split on non-alphanumeric?

- And have predictable, regular forms
  - Normalisation
    - `WINDOWS = Windows = windows`
    - First try: convert to lower-case?

Regular Expressions

- Disjunctions `[wW] (windows|Windows)`
- Ranges `[a-z] [A-Z] [0-9] [a-zA-Z]`
- Negations `[^a]`
- Wildcards `? . * +`

  `windows?` `window.?` `window*`
- Anchors `^ $`
- Classes `\w \s \W \S`
- Find all instances of the word "the":

  `the` Misses capitalized examples

  `[tT]he` Incorrectly returns `other` or `theology`

  `[^a-zA-Z][tT]he[^a-zA-Z]`

  `\b[tT]he\b`
- Surprisingly powerful and commonly used!

Tokenisation

- For English:

  `split(r'\W')` or `split(r'\b')`

  – (or many similar options)
- How about Chinese?

  莎拉波娃现在居住在美国东南部的佛罗里达。

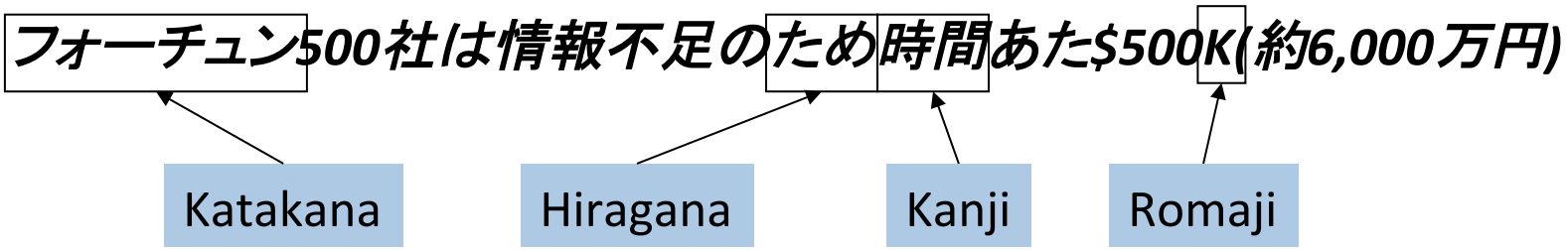
  莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达

  Sharapova now lives in US southeastern Florida

  – Characters are generally 1 syllable and 1 morpheme.
  – Average word is 2.4 characters long. No spaces!
- Standard baseline segmentation algorithm:
  1. Start pointer at beginning of string
  2. Find longest word in dictionary matching string starting at pointer
  3. Move pointer over word, go to 2

Tokenization

- How about Japanese?
  - multiple alphabets intermingled
  - dates/amounts in multiple formats



End-user can express query entirely in hiragana!

- How about German?
  - Lebensversicherungsgesellschaftsangestellter
  - 'life insurance company employee'
  - (wait for sequence modelling lecture …)

Spelling Correction

- Spelling errors are common (26% of web queries)
  - Detection
  - Correction
- Hard to estimate error probability directly
  - (have we seen x before?) $P(w \mid x)$
- Bayes' Rule again!

$$P(w \mid x) = \frac{P(x \mid w)P(w)}{P(x)}$$

Spelling Correction

- Need probability of original word
  - estimate from corpus
  - "unigram language model" (see next week)

$$P(w)$$

- Need probability of error introduction
  - "noisy channel model"
  - e.g. probability of:
    - Leaving a character out
    - Tranposing two characters
    - Adding a space character
    - … etc
  - estimate from corpus of errors

$$P(x \mid w)$$

- E.g. Kernighan et al (1990)

$$P(x|w) = \begin{cases} \frac{\text{del}[w_{i-1}, w_i]}{\text{count}[w_{i-1} w_i]}, & \text{if deletion} \\ \frac{\text{ins}[w_{i-1}, x_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

Stemming / Lemmatisation

- We often want base form (stem, lemma)

  `windows → window`

  `estimating → estimate (→ estimat)`

  `running → run`

  `are, is → be`

- **Morphemes**:
  - The small meaningful units that make up words
  - **Stems**: The core meaning-bearing units
  - **Affixes**: Bits and pieces that adhere to stems
    - Often with grammatical functions

Porter's algorithm
The most common English stemmer

Step 1a

sses	→ ss	caresses	→ caress
ies	→ i	ponies	→ poni
ss	→ ss	caress	→ caress
s	→ ø	cats	→ cat

Step 1b

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

Step 2 (for long stems)

ational	→ ate	relational	→ relate
izer	→ ize	digitizer	→ digitize
ator	→ ate	operator	→ operate
…			

Step 3 (for longer stems)

al	→ ø	revival	→ reviv
able	→ ø	adjustable	→ adjust
ate	→ ø	activate	→ activ
…			

Why check for vowels in Step 1b?

(*v*)ing	→ ø	walking	→ walk
		sing	→ sing

Stemming in other languages

- What if we're working with Turkish?
  - Uygarlaştıramadıklarımızdanmışsınızcasına
  - `(behaving) as if you are among those whom we could not civilize'
  - Uygar `civilized' + laş `become'
    + tır `cause' + ama `not able'
    + dık `past' + lar 'plural'
    + ımız 'p1pl' + dan 'abl'
    + mış 'past' + sınız '2pl' + casına 'as if'

- More advanced grammar-based methods:
  - e.g. finite state transducers (see later)

Advanced Features

- Now we can extract word features
  - unigrams, n-grams

    `milk is good and not expensive`

    `[START milk], [milk is], [is good], [good and], [and not], [not expensive], [expsnsive END]`

    `[START milk is], [milk is good], [is good and], [good and not], …`
  - What about other features?
- Word class features

  `I like milk, I like sugar, … → I like NN`
  - Needs part-of-speech tagging
- Semantic roles

  `I like milk, I like sugar, … → like(me,NN)`
  - Needs syntactic/semantic parsing

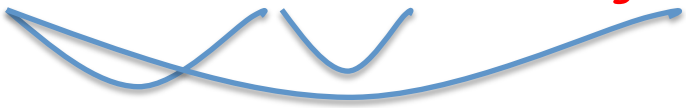
Negation

- What do n-gram models do with:

  `milk is not very good`
  `milk is not really very good`
  `milk is not bad but good`

- Really we need to know syntactic/semantic dependencies
  - Parsing: see later lectures

    `milk is not bad but good`



- But we can use some handy approximations
  - e.g. add neg_ prefix to every word between negation and punctuation / coordination

    `milk is not neg_really neg_very neg_good`
    `milk is not neg_bad but good`

Evaluation

- How do we evaluate?

- What if we have unbalanced classes?
  - E.g. 90% negative, 10% positive
  - How good is the accuracy of this classifier?

    x → negative

Precision and recall

- **Precision**: % of selected items that are correct
  Recall: % of correct items that are selected

	correct	not correct
selected	tp	fp
not selected	fn	tn

$P = tp/(tp+fp)$

$R = tp/(tp+fn)$

A combined measure: F

- A combined measure that assesses the P/R tradeoff is F measure (weighted harmonic mean):

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

- The harmonic mean is a very conservative average

- People usually use balanced F1 measure
  - i.e., with $\beta = 1$ (that is, $\alpha = ½$): $F = 2PR/(P+R)$

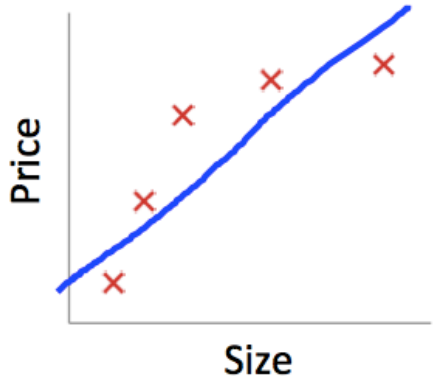
Unbalanced Datasets

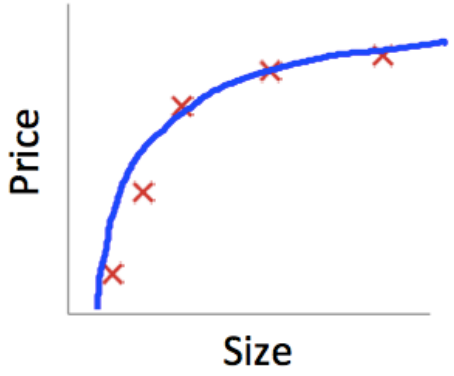
- What if our training data isn't balanced?

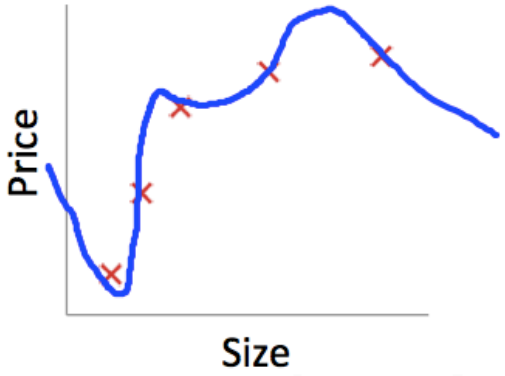
$$= \underset{c \in C}{\operatorname{argmax}} P(d \mid c) P(c)$$

- (and logistic example)
- Is this representative of the test data?
  - If no; probably not OK!
  - If yes; probably OK for NB, maybe not for LR
- Solutions:
  - over/under-sample training data
    - (beware over-sampling in cross-validation!)
  - weighted cost function
    - (usually better if possible)

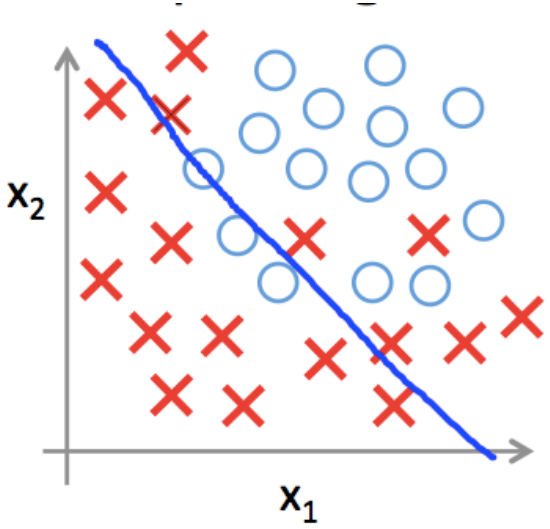
Overfitting

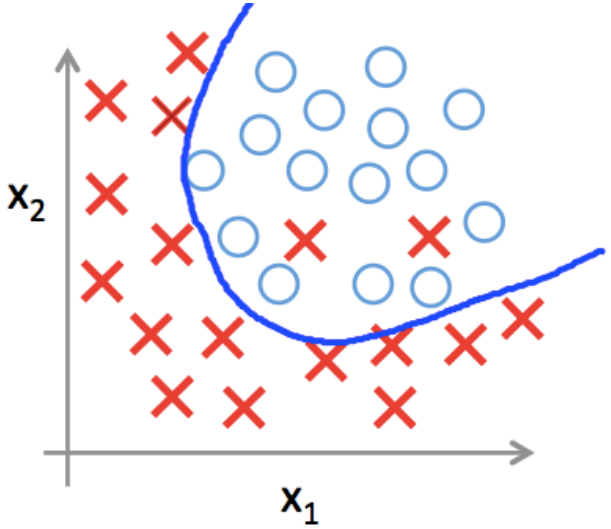


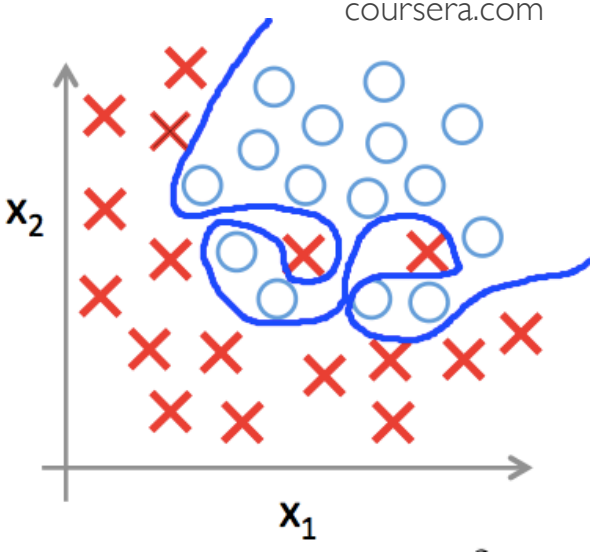




Andrew Ng, coursera.com

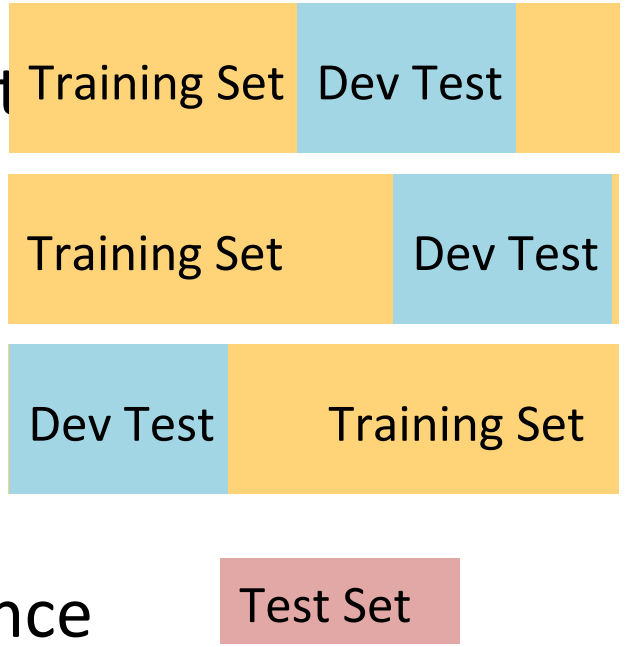






Development Test Sets and Cross-validation

- Metric: P/R/F1 or Accuracy
- Unseen test set
  - avoid overfitting ('tuning to the test
  - more conservative estimate of performance
- Cross-validation over multiple splits
  - Handle sampling errors from different datasets
  - Pool results over each split
  - Compute pooled dev set performance

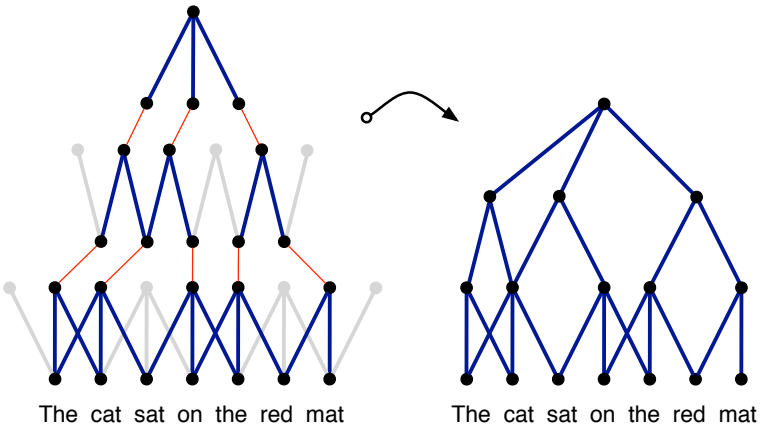


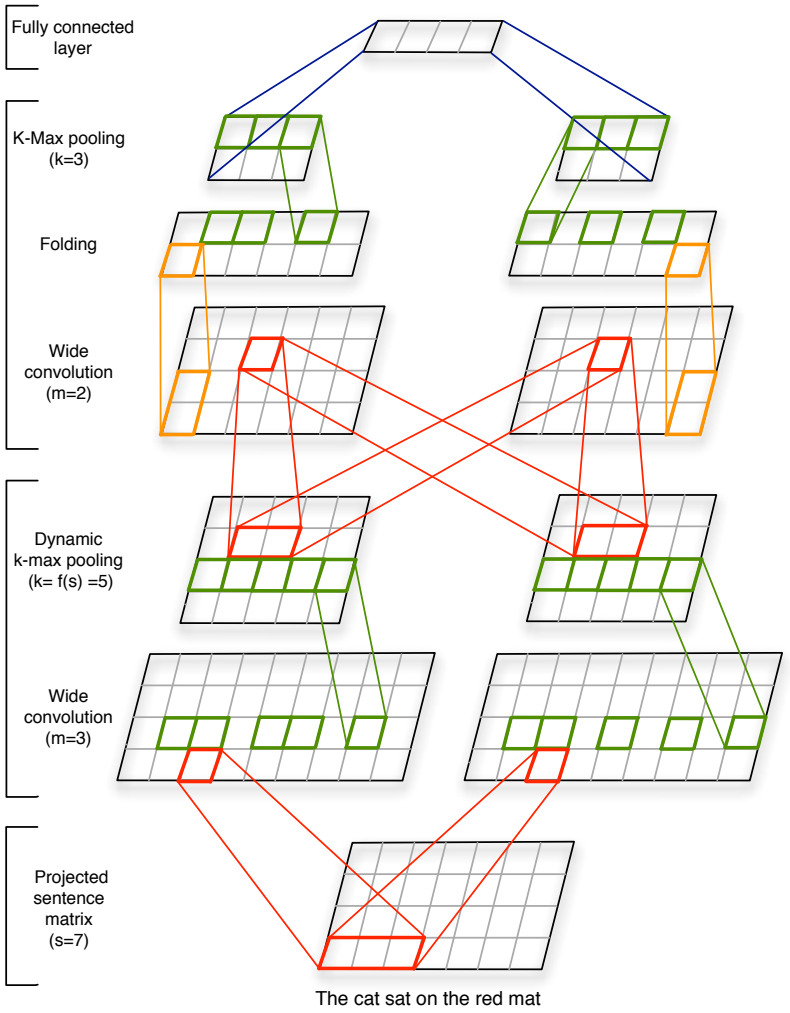
Avoiding over-fitting

- Always use a held-out test set
- Select parameters via cross-validation
  - (be careful with selection criteria!)
  - Or with a separate development set
- Use regularisation / smoothing
- Avoid high dimensionality of feature set
  - N >> d
  - You know about feature selection from Data Mining …

Neural Networks

e.g. Kalchbrenner et al. (2014)





Example: Sentiment Analysis

- Purver et al. (2012)
  - See paper on QM+

Experimenting with Distant Supervision for Emotion Classification

Matthew Purver[*] and **Stuart Battersby**[†]
[*]Interaction Media and Communication Group [†]Chatterbox Analytics
School of Electronic Engineering and Computer Science
Queen Mary University of London
Mile End Road, London E1 4NS, UK
m.purver@qmul.ac.uk stuart@cbanalytics.co.uk

Abstract

We describe a set of experiments using automatically labelled data to train supervised classifiers for multi-class emotion detection in Twitter messages with no manual intervention. By cross-validating between mod-

in unconventional style and without accompanying metadata, audio/video signals or access to the author for disambiguation, how can we easily produce a gold-standard labelling for training and/or for evaluation and test? One possible solution that is becoming popular is crowd-sourcing the la-

Choices

- Training set?
- Labelling?
- Feature selection?
- Test set?
- Features?
- Classifier?
- Balancing?

Live London Emotion Map

http://www.eecs.qmul.ac.uk/~mpurver/emomap/

- (happy = yellow, sad = blue, angry = red)