Queen Mary
University of London

ECS763
Natural Language Processing

Unit 4: Text Classification II: Discriminative models

Lecturer: Julian Hough
School of Electronic Engineering and Computer Science

OUTLINE

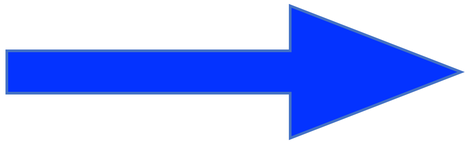
1) General feature extraction functions for text classification
2) Feature selection and weighing methods
3) Going discriminative: Logistic Regression
4) Alternative linear classifier: SVMs

OUTLINE

1) General feature extraction functions for text classification
2) Feature selection and weighing methods
3) Going discriminative: Logistic Regression
4) Alternative linear classifier: SVMs

Text Classification

```
i love @justinbieber #sarcasm
```

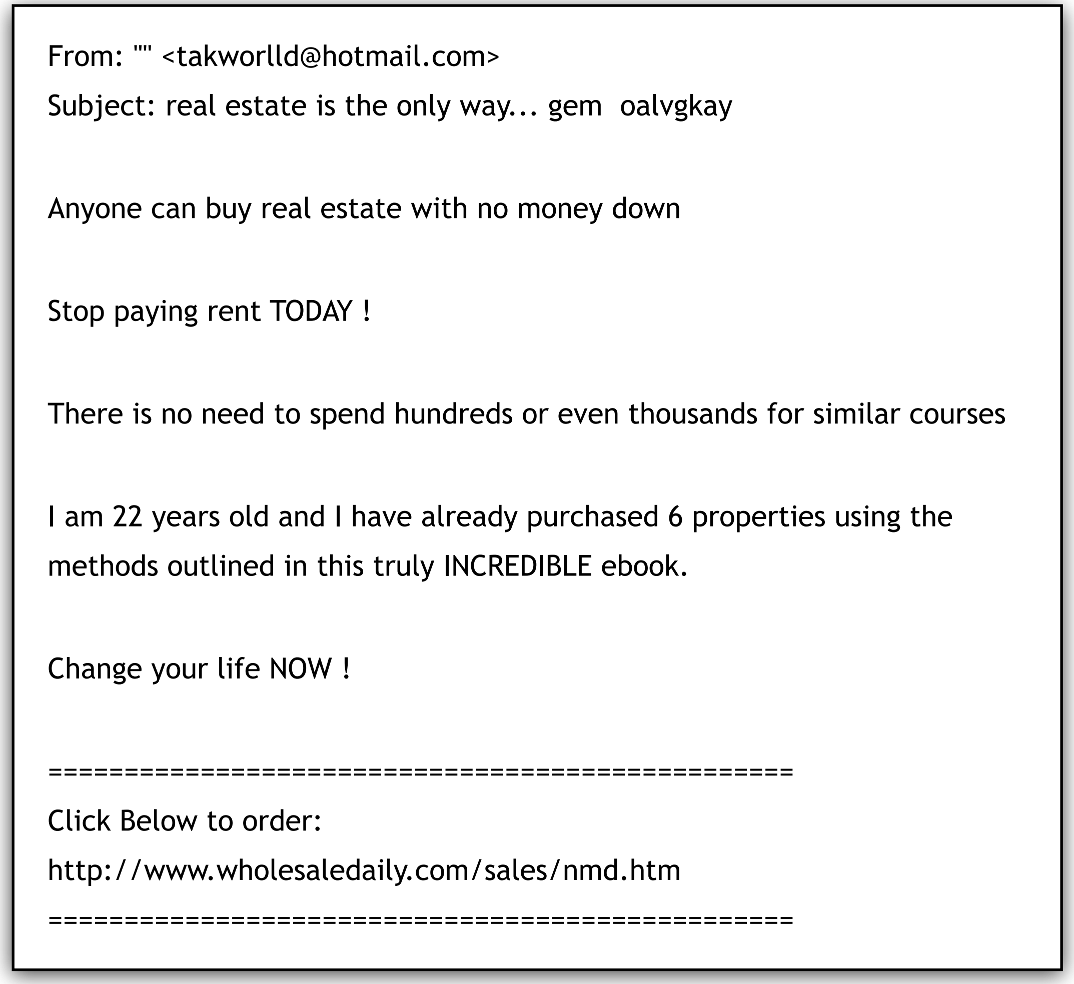


Positive

or

Negative?

Text Classification



From: "" <takworlld@hotmail.com>

Subject: real estate is the only way... gem oalvgkay

Anyone can buy real estate with no money down

Stop paying rent TODAY !

There is no need to spend hundreds or even thousands for similar courses

I am 22 years old and I have already purchased 6 properties using the methods outlined in this truly INCREDIBLE ebook.

Change your life NOW !

==

Click Below to order:

http://www.wholesaledaily.com/sales/nmd.htm

==

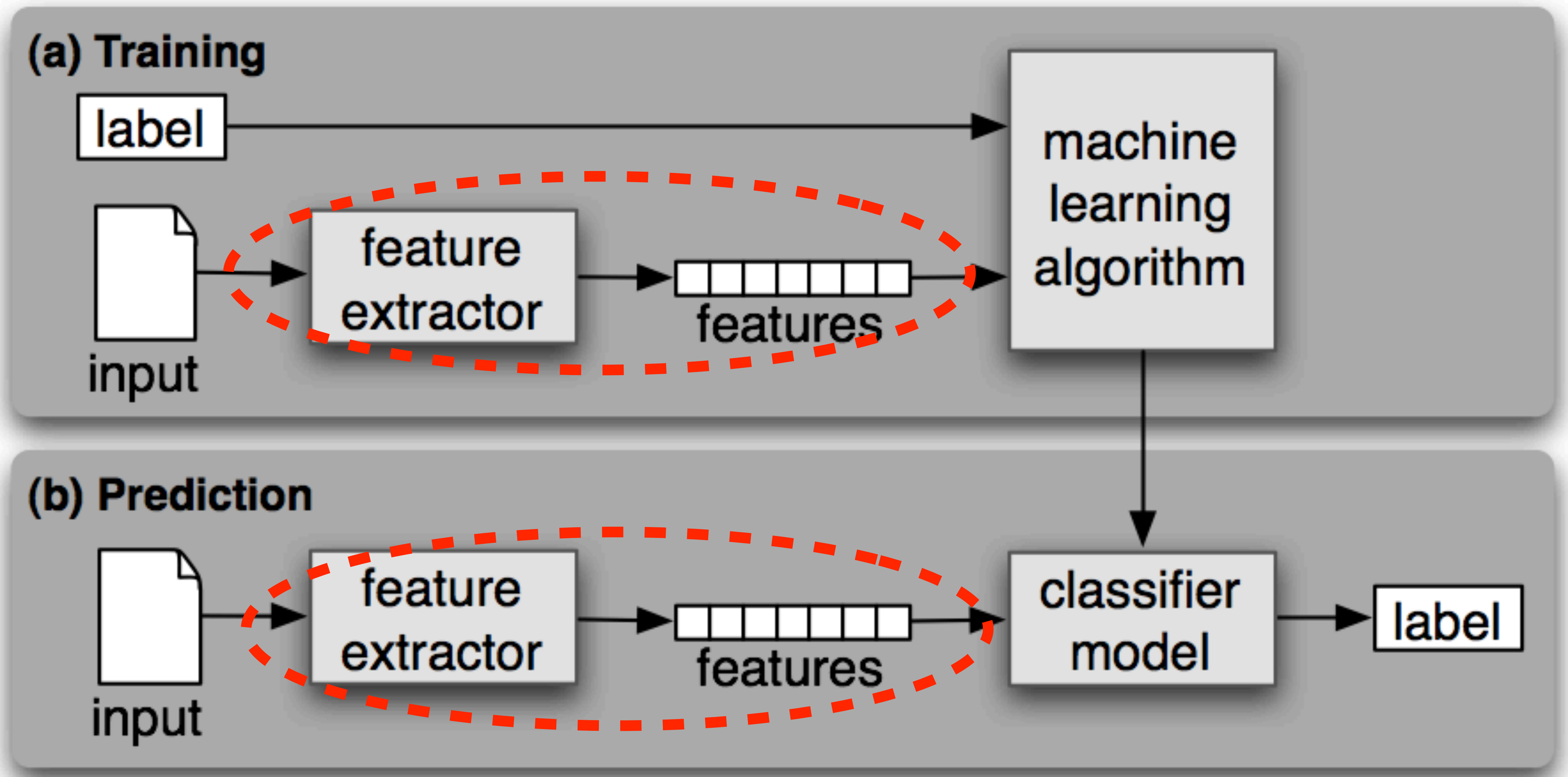
Spam

or

not Spam?

Text Classification: definition

- *Input*:
  - a document x
    - Issue: how to represent text documents
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
- *Output*: a predicted class $c \in C$
- *Classifier:* a function $f : x \rightarrow c$
- *Examples*: spam filtering, sentiment analysis
  - C = {*spam, not_spam*}
  - C = {*positive, negative*}
  - C = {*angry, happy, sad, excited, confused….* }


(a) Training
label
machine learning algorithm
input
feature extractor
features
(b) Prediction
input
feature extractor
features
classifier model
label

Vectorizing inputs according to features

- We can characterise each instance/document in terms of a list of feature-value pairs.
- These in practice are represented to a classifier model as a **vector** *x* where the **basis** of each vector is the same for each document, e.g. for a bag-of-words model this is just the **vocabulary** (e.g. in this model the count for ‘Mo’ for a document will always be in position *2*) :

$$x = [\underset{\textit{I}}{0}, \underset{\textit{Mo}}{3}, \underset{\textit{Bill}}{0}, \underset{\textit{Mary}}{0}, \underset{\textit{John}}{0}, \underset{\ldots}{\ldots}, \underset{\textit{love}}{2}, \underset{\textit{like}}{0}]$$

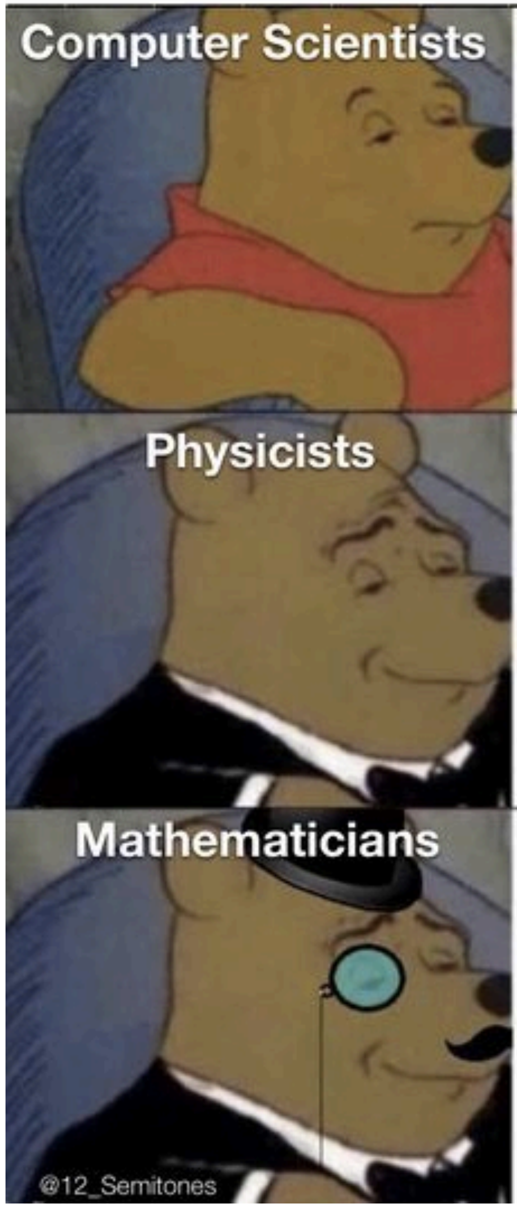
Vectorizing inputs according to features

- For bag-of-words models these vectors are **sparse** - lots of 0 counts!

- In practice in implementations we don’t always have to populate all vectors ourselves and we just supply positive values.

- e.g. Python `sklearn` models have `CountVectorizer` objects which do this for us)

Definition of a vector:



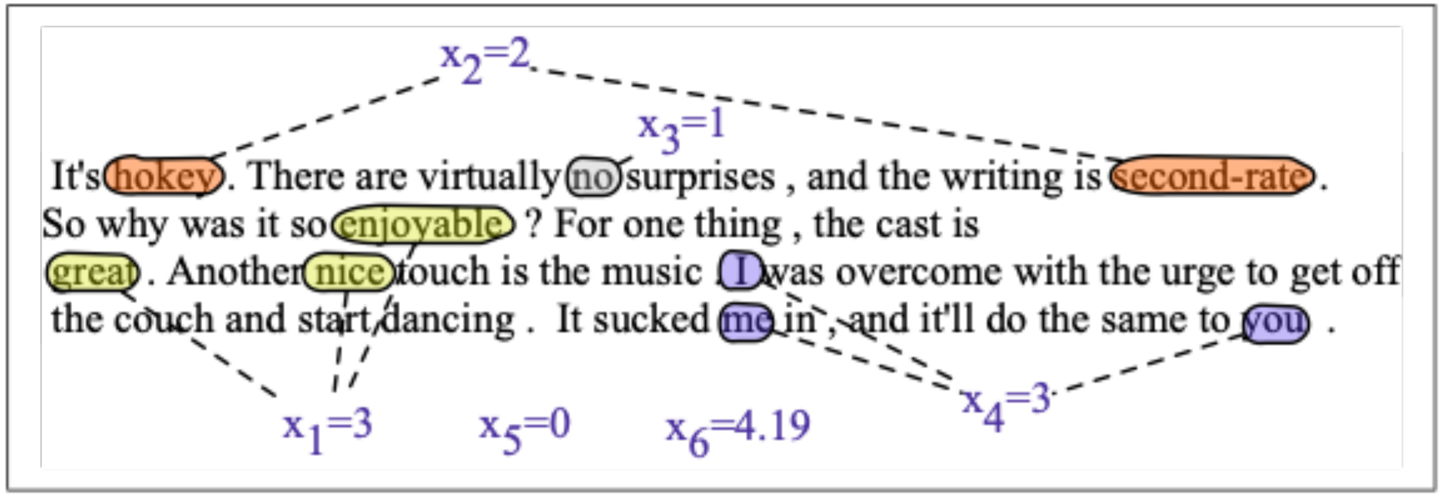
It’s an array.

It’s a quantity with magnitude and direction.

It’s an element of a vector space, which is a set of objects that follow certain closure properties and axioms under vector addition and scalar multiplication.

Vectorizing inputs according to features

- In reality, the feature set can be **anything quantifiable (inc binary)**, but we still represent instances in terms of a vectorised version of it consistently:



It's hokey. There are virtually no surprises , and the writing is second-rate. So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if ``no''} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if ``!''} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Features for text classification

- Bag of words assumption
  - Good baseline, often good to keep them there along with other features.
- N-gram features
  - Subsequences of N consecutive words
  - Often with START, END symbols
  - E.g. for n = 2 (bigrams)

```
<start> hi matt want some viagra? <end>

<start> hi
hi matt
matt want
want some
some viagra
viagra <end>
```

- Linguistic features: part-of-speech tags, parse tree features (later in the module).

Deception detection using NLP methods

- Create a corpus of deceptive and non-deceptive text.
- Train a classifier using the corpus.
- Typically, classifiers use STYLOMETRIC features.

Stylometry

- Assumes that the essence of the **individual style** of an author can be captured with reference to a number of quantitative criteria, called **discriminators.**
- Obviously, some (many) aspects of style are conscious and deliberate:
  - As such they can be easily imitated and indeed often are.
  - Many famous pastiches, either humorous or as a sort of homage.
- Computational stylometry is focused on **subconscious** elements of style less easy to imitate or falsify.

Basic methodologies

- Word or sentence length too obvious and easy to manipulate.
- Frequencies of letter pairs strangely successful, though limited.
- Distribution of words of a given length (in syllables), especially **relative frequencies**, ie length of gaps between words of same syllable length.

Linguistic features

- Basic Measurements:
  - Average syllable/word/sentence count, letter distribution, punctuation.
- Lexical Density
  - *Unique_Words / Total_Words*
- Gunning-Fog Readability Index:
  - *0.4 * (Average_Sentence_Length + 100 * Complex_Word_Ratio)*
  - Result: years of formal education required to read the text.

OUTLINE

1) General feature extraction functions for text classification
2) Feature selection and weighing methods
3) Going discriminative: Logistic Regression
4) Alternative linear classifier: SVMs

The need for feature selection

- Text collections have a large number of features
  - 10,000 – 1,000,000 unique words – and more
- Feature selection can make using a particular classifier feasible
  - Some classifiers can’t deal with 100,000s of features
- It can also reduce training time
  - Training time for some methods is quadratic or worse in the number of features (e.g., logistic regression)
- Improve generalization
  - Eliminate noise features
  - Avoid overfitting

Feature set reduction via preprocessing using linguistic knowledge

- Ways of reducing feature space for text using info about words through sensible **pre-processing**:
  - **Stemming**
    - Laugh, laughs, laughing, laughed -> laugh
  - **Stop word removal**
    - E.g., eliminate all prepositions (’to’, ‘in’)/very frequent functional words
  - **Conversion to lower case**
  - **Tokenization**
    - Break on all special characters: fire-fighter -> fire, fighter
    - What about social media conventions like emojis? :) :(;)
  - **Spelling correction**

- It does depend on the application: a craft skill/matter of domain relevant investigation!

Feature Selection using statistical ‘weighing’ methods

- For many, if not most applications, it is better to **weigh** terms with respect to a document instead of simply count their frequencies
- Yang and Pedersen 1997: a comparison of different selection criteria
  - DF – document frequency
  - IG – information gain
  - MI – mutual information
  - CHI – chi square
- Other widely used methods
  - TF/IDF (theory later in the module)
- Common strategy
  - Compute statistic for each term
  - Keep n terms with highest value of this statistic

Feature selection via Document Frequency

- Number of documents/instances a term occurs in
- Is sometimes used for eliminating both very frequent and very infrequent terms, to define a restricted vocabulary in terms of:
  - **Minimum document frequency**
    - E.g. for a min. doc. freq of 2, a word must appear in at least 2 documents for it to be included in the feature set.
  - **Maximum document frequency**
    - E.g. for a max. doc. freq of 10, the word must not appear in more than 10 documents for it to be included.

Feature selection via Information Theoretic measures

- We need notions from **Information Theory** (Shannon 1948) for effective feature selection:
  - **Mutual Information**
  - **Information Gain**
  - **Entropy**

Feature selection via Mutual Information

- We might not want to use all words, but just reliable, good discriminators
- In training set, choose *k* words which best discriminate the categories.
- One way is in terms of **(pointwise) Mutual Information ((P)MI)**

(Pointwise) Mutual Information

$$I(x, c) = log\frac{p(x \wedge c)}{p(x) \times p(c)}$$

$$I_{avg}(x) = \sum_{i=1}^{m} p(c_i)I(x, c_i)$$

Feature selection via MI (contd.)

- For each category we build a list of *k* most discriminating terms.
- For example (on 20 Newsgroups):
  - ***sci.electronics:*** circuit, voltage, amp, ground, copy, battery, electronics, cooling, …
  - ***rec.autos:*** car, cars, engine, ford, dealer, mustang, oil, collision, autos, tires, toyota, …
- Greedy: does not account for correlations between terms
- In general feature selection is *necessary* for binomial NB, but not for multinomial NB

Information Gain

- A metric much used in ML to decide on the importance of features e.g., when learning decision trees
- It measures how much a given feature (word) *x* REDUCES THE ENTROPY

Entropy

Information theory (Shannon, 1948):

- **Entropy –** amount of uncertainty in a distribution
- **Minimum number of binary questions (bits) required to answer a 'question'**- the fewer, the less uncertainty
    - e.g. Resolving whether a **heads or tail is thrown** requires only **1 bit** of information (one question to be asked- Heads or Tails?).

$$H(c) = -\sum_{i=1}^{m} p(c_i) log(p(c_i))$$

Information Gain

$$
\begin{aligned}
G(x) = &-\sum_{i=1}^{m} p(c_i)log(p(c_i)) \\
&+p(x)\sum_{i=1}^{m} p(c_i|x)log(p(c_i|x)) \\
&+p(\neg x)\sum_{i=1}^{m} p(c_i|\neg x)log(p(c_i|\neg x))
\end{aligned}
$$

Tip: Avoiding over-fitting on training data

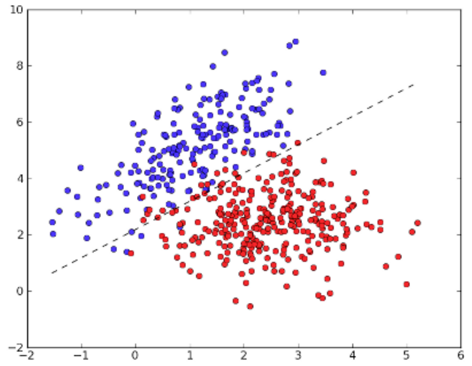
- Always use separate test set in development - don’t test on training set!
- Don’t use test set for **hyper-parameter** selection, use development set!, e.g.:
  - Range of n for ngrams 1,2,3… etc.
  - Smoothing method for Naive Bayes/regularisation.
- Avoid too high a dimensionality of feature set which can ‘memorise’ the training data:
  - N > d
  - Techniques for feature reduction include minimum document frequency, ranking with information gain, mutual information and other measures to select the top *k* features.

OUTLINE

1) General feature extraction functions for text classification
2) Feature selection and weighing methods
3) Going discriminative: Logistic Regression
4) Alternative linear classifier: SVMs

Discriminative approaches

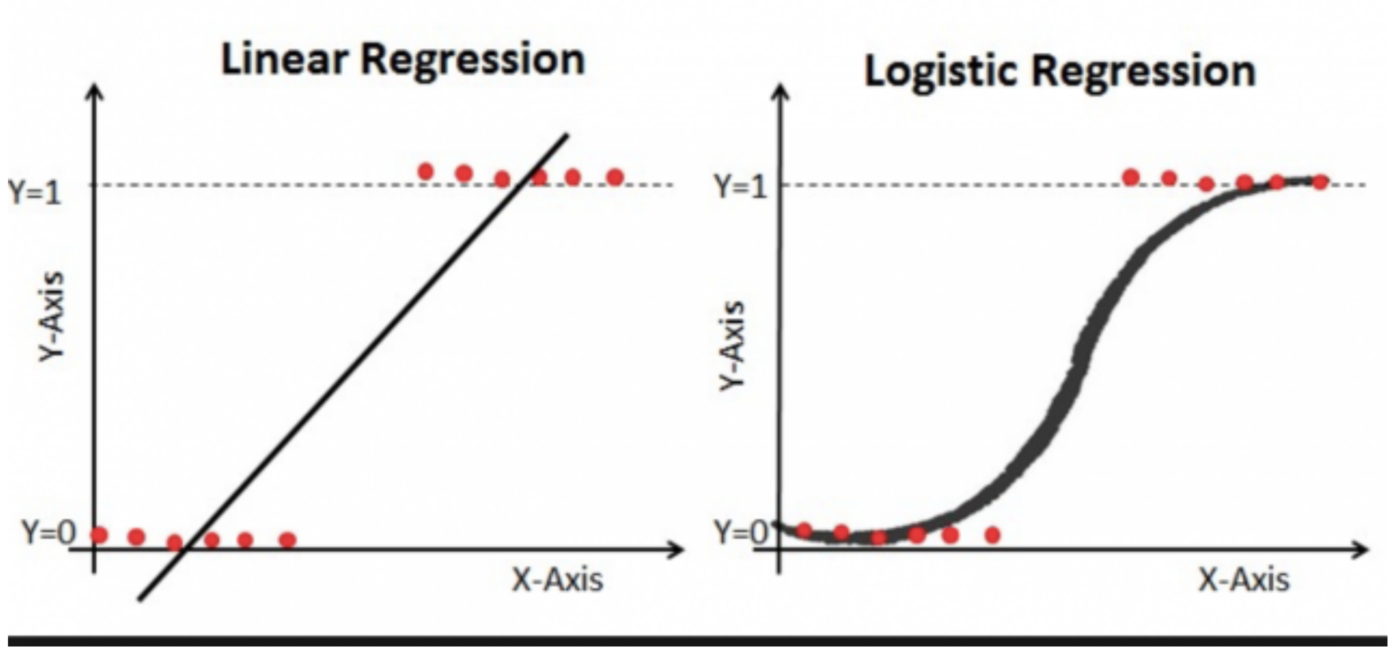
- Naïve Bayes is a linear classifier
  - Decision is a linear function of weighted feature values
- But:
  - we calculate feature weights & boundary independently
    - (i.e. we're assuming statistical independence)
  - we're using a **generative** model



- Why not:
  - estimate the boundary **directly** in the data?
  - use a **discriminative** model?

Logistic Regression

- We can use a **discriminative nonlinear classifier** like **Logistic Regression**.



Logistic Regression

Naive Bayes

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \ \overbrace{P(d|c)}^{\text{likelihood}} \ \overbrace{P(c)}^{\text{prior}}$$

Logistic Regression

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \ \overset{\text{posterior}}{P(c|d)}$$

Logistic Regression

- Classification with logistic regression:

$$\hat{y} = \underset{y \in Y}{\text{argmax}}\ p(y \mid x)$$

- Where x is an instance of data (e.g. a document), represented as a vector/array, representing, e.g. a bag of words count for that document where the element at each index x_i corresponds to a feature, e.g. the count for a particular word in the doc:

$$x = [\underset{\textit{I}}{0},\ \underset{\textit{Mo}}{3},\ \underset{\textit{Bill}}{0},\ \underset{\textit{Mary}}{0},\ \underset{\textit{John}}{0},\ \underset{\cdots}{\ldots},\ \underset{\textit{love}}{2},\ \underset{\textit{like}}{0}]$$

- And p(ylx) is a scalar between [0,1]

Logistic Regression

- When we're doing binary classification we can just think of the true label we want as being 1 or 0
  - e.g. Positive (1) vs Negative (0) in sentiment analysis
  - e.g. Fake (1) vs not-fake (0) in spam detection.

- To classify an instance, we apply a weight matrix *w* to the input vector *x* by multiplication, linearly combining the features, and add a bias term *b.* Note at this stage this might still not be a proper probability between [0,1] so is not yet p(y=1|x):

$$Z = (\sum_{i=1}^{n} w_i x_i) + b$$

$$Z = w \cdot x + b$$

Logistic Regression

- Each of the weights w_i in w in a binary classification is a **coefficient** between the feature x_i and the positive class.

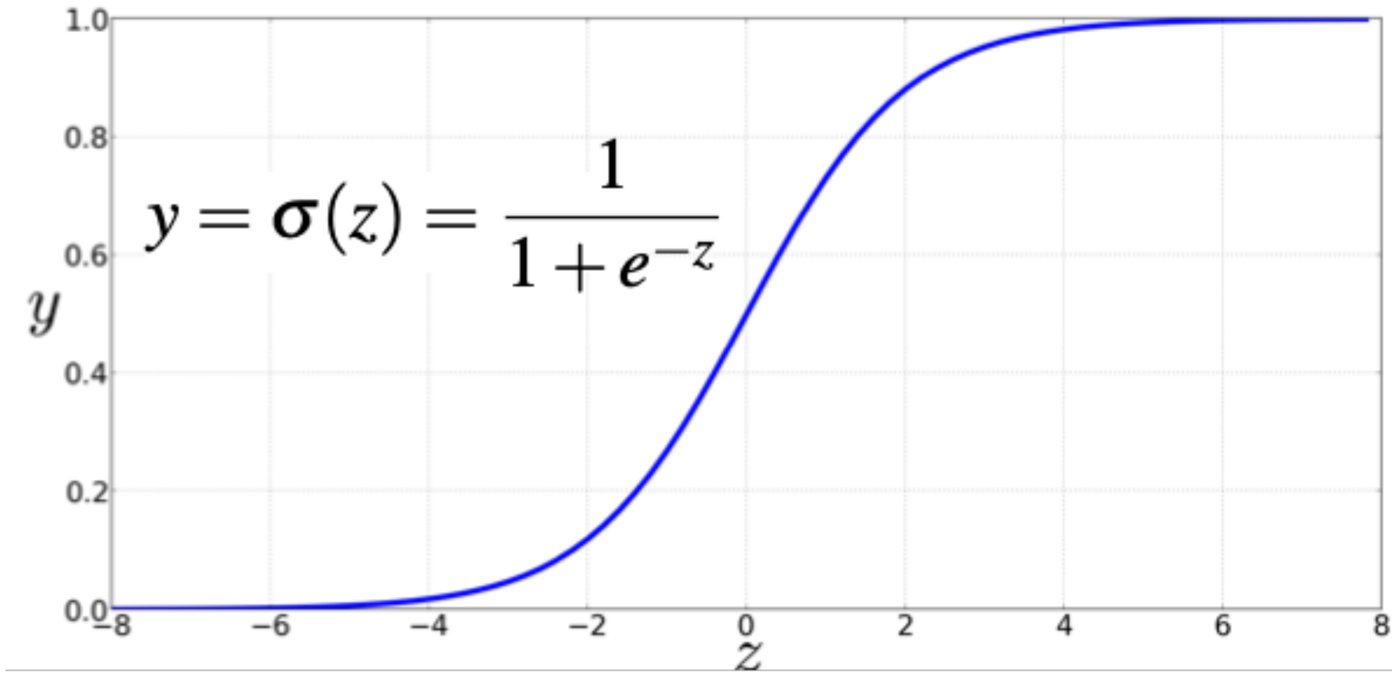
- For feature x_i, weight w_i tells is how important x_i is for predicting y=1. e.g. where 1= positive for reviews:
    - x_{12} ="review contains ‘`awesome`’": w_{12} = `+10`
    - x_{18} ="review contains ‘`abysmal`’": w_{18} = `-10`
    - x_{256} =“review contains ‘`mediocre`’": w_{256} = `-2`

- Therefore the cross product with the intercept/bias *b*, i.e.: $z = w \cdot x + b$ will give a high value if the features have higher coefficients, and low value if they are lower.

Logistic Regression

- The problem is z will **not be a proper probability value**- solution is to add a **sigmoid** function over *z* to make it between [0,1].

$$y = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\exp(-z)}$$



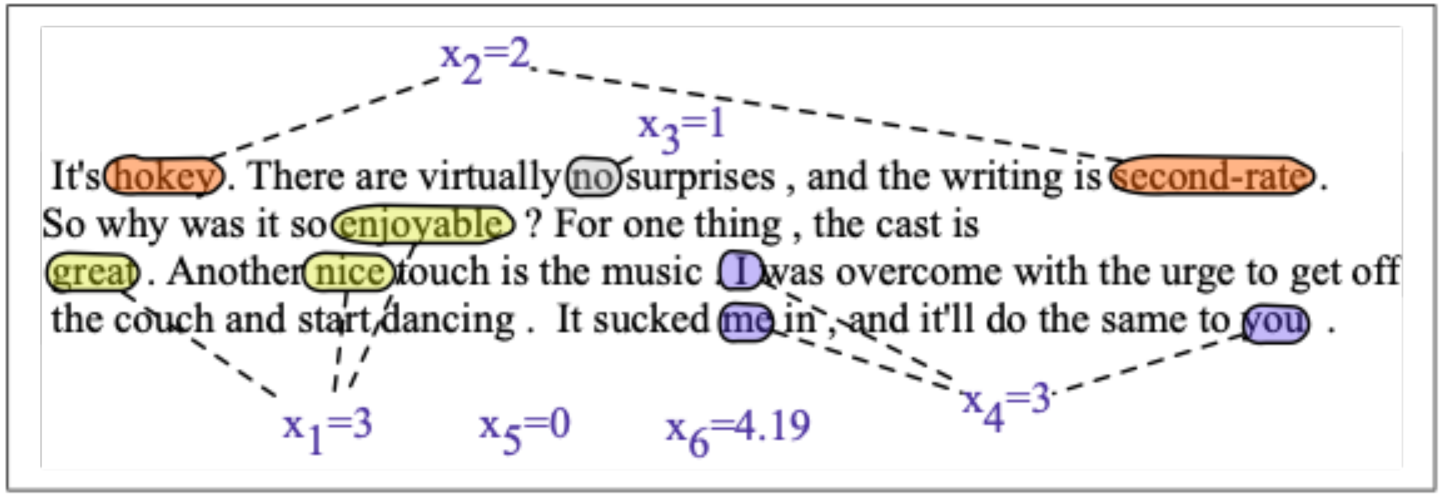
Logistic Regression

- $\sigma(z)$ is now the approximation to p(y=1|x), and given that real value, in binary classification we make a classification based on a threshold or **decision boundary** (e.g. 0.5)

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Logistic Regression

- Example: Sentiment analysis (positive + vs - negative)
- Text instance:


It's hokey. There are virtually no surprises , and the writing is second-rate. So why was it so enjoyable ? For one thing , the cast is great. Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

- Feature values x:

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Logistic Regression

- Example: Sentiment analysis (positive + vs - negative)
- Suppose our model is (through training): *w* = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] and *b* = 0.1

$$Z = (\sum_{i=1}^{n} w_i x_i) + b$$

- What's the likelihood of y=+ for input x = [3, 2, 1, 3, 0, 4.19] ?:

$$\begin{aligned} p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= \sigma(.833) \\ &= 0.70 \end{aligned} \tag{5.6}$$

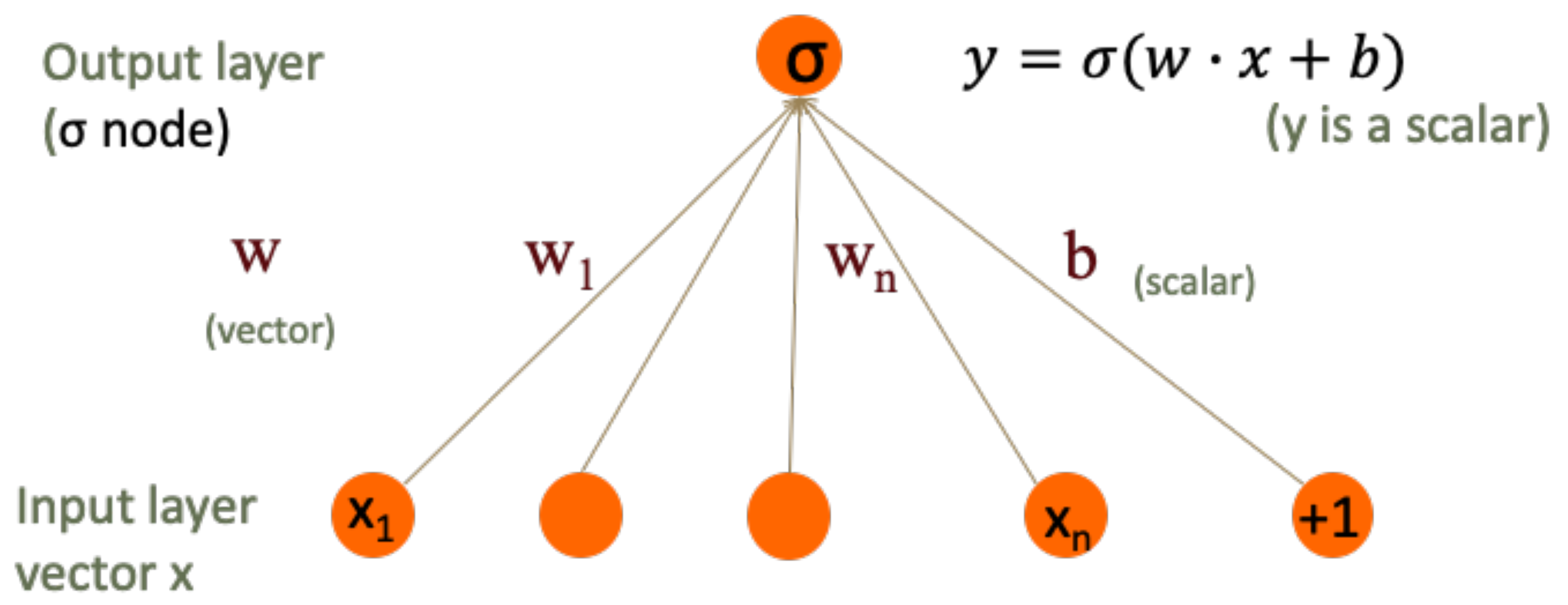
- What's the likelihood of y=- for same input?:

$$\begin{aligned} p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\ &= 0.30 \end{aligned}$$

Logistic Regression

Binary Logistic Regression as a 1-layer Network

(we don't count the input layer in counting layers!)



Multinomial Logistic Regression

- What if we want to go beyond binary classification to multi-class classification?
- We need **multinomial logistic regression.**
- The probability of everything must still sum to 1

$$P(\text{positive}|\text{doc}) + P(\text{negative}|\text{doc}) + P(\text{neutral}|\text{doc}) = 1$$

Multinomial Logistic Regression

- For multinomial LR we use a generalization of the sigmoid called the **softmax:**
  - Takes a vector z = [z_1,z_2 ..., z_k] of *k* arbitrary values (corresponding to *k* classes)
  - Outputs a probability distribution over all *k* classes
  - each value in the range [0,1].
  - all the values summing to 1.

- The probability estimation for a given class *i* is:

$$\text{softmax}(z_i) = \frac{\text{exp}(z_i)}{\Sigma_{j=1}^{k}\text{exp}(z_j)} 1 \leq i \leq k$$

Multinomial Logistic Regression

- The probability estimation for a given class *i* is:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\Sigma_{j=1}^{k} \exp(z_j)} 1 \leq i \leq k$$

- The denominator $\Sigma_{j=1}^{k} \exp(z_j)$ is used to normalise all the values into probabilities in a proper distribution:

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\Sigma_{j=1}^{k} \exp(z_j)}, \frac{\exp(z_2)}{\Sigma_{j=1}^{k} \exp(z_j)}, \ldots, \frac{\exp(z_k)}{\Sigma_{j=1}^{k} \exp(z_j)} \right]$$

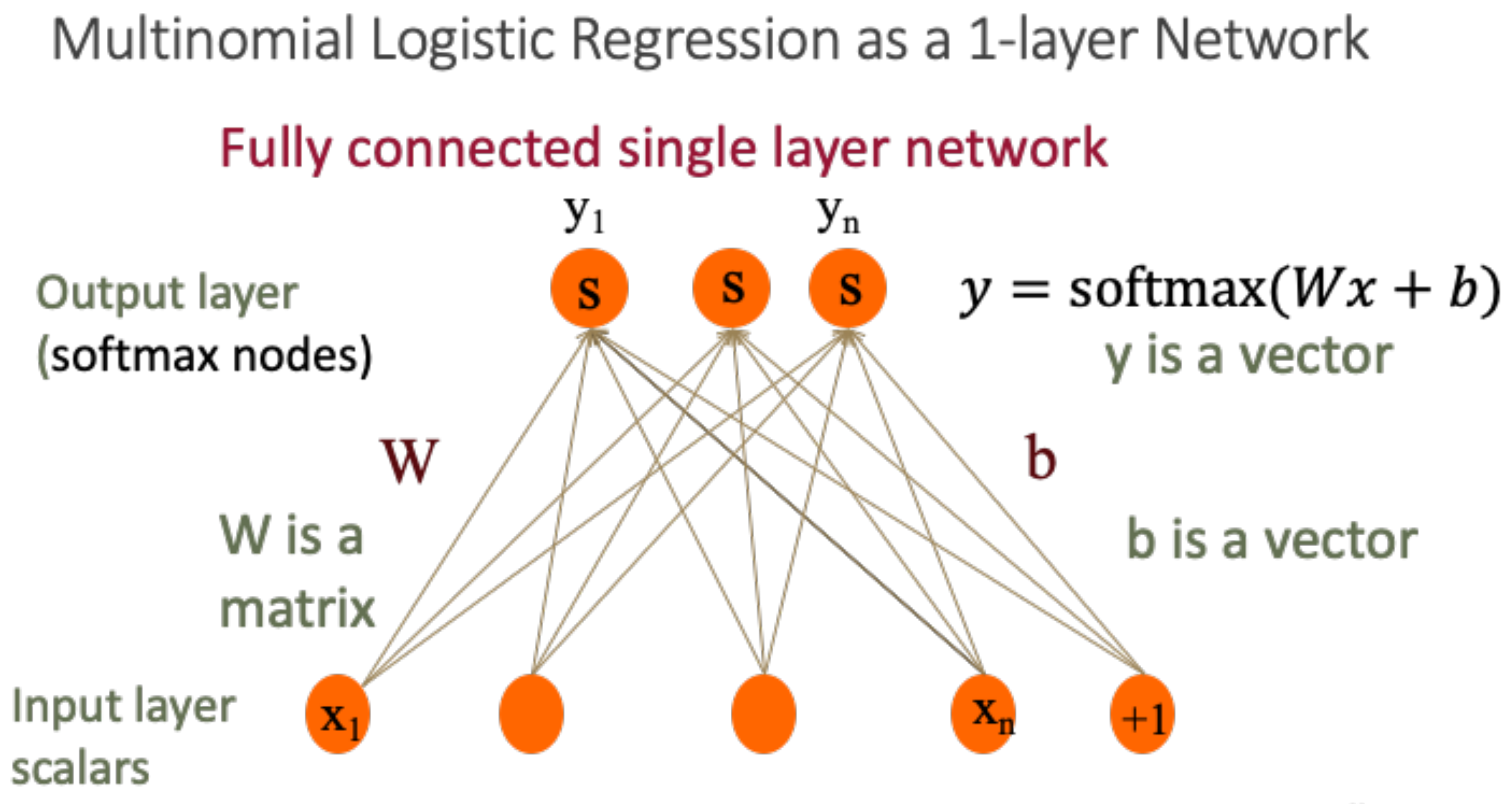
Multinomial Logistic Regression

- Softmax in multinomial logistic regression in terms of vector multiplication. For a given class *c* for an input *x*, this is:

$$p(y = c \,|\, x) = \frac{\text{exp}(w_c \cdot x + b)}{\Sigma_{j=1}^{k} \text{exp}(w_j \cdot x + b)}$$

- Input is still the dot product between the weight vector *w* and the input vector *x* (for a specific binary class *c*).
- But now we need separate weight vectors w_j for each of the *k* classes.

Multinomial Logistic Regression


Multinomial Logistic Regression as a 1-layer Network
Fully connected single layer network
y_1
y_n
Output layer
(softmax nodes)
s
s
s
$y = \text{softmax}(Wx + b)$
y is a vector
W
b
W is a
matrix
b is a vector
Input layer
scalars
x_1
x_n
+1

Multinomial Logistic Regression

Features in binary versus multinomial logistic regression

Binary: positive weight → y=1 neg weight → y=0

$$x_5 = \begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases} \qquad w_5 = 3.0$$

Multinominal: separate weights for each class:

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	-5.3

Learning the weights

Where do the *w*'s come from?

Supervised classification:

- We know the correct label *y* (either 0 or 1) for each *x*.
- But what the system produces is an estimate, $\hat{y}$
- We want to set *w* and *b* to minimize the distance between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.
- We need a distance estimator: a **loss function** or a **cost function**
- We need an **optimization algorithm** to update *w* and *b* to minimize the loss until good enough/convergence.

Learning the weights

Where do the *w*’s come from?

- Loss function: **cross-entropy**
- Optimization algorithm: **gradient descent**

Loss functions

- We want to know how far is the classifier output:

  $\hat{y} = \sigma(w \cdot x+b)$

- from the true output:

  y [= either 0 or 1]

- We'll call this difference:

  $L(\hat{y}, y)$ = how much $\hat{y}$ differs from the true y

Cross-entropy loss

- Goal: maximize probability of the correct label *p*(*y*|*x*) through the exponential function:
- Maximize:

$$p(y|x) = \hat{y}^{y}\,(1-\hat{y})^{1-y}$$

- Now flip sign to turn this into a loss: something to **minimize**
- **Cross-entropy loss** between $\hat{y}$ and y

- Minimize:

$$L_{\mathrm{CE}}(\hat{y},y) = -\log p(y|x) = -[y\log\hat{y} + (1-y)\log(1-\hat{y})]$$

- Or, plugging in definition of $\hat{y}$:

$$L_{CE}(\hat{y},y) = -\left[y\ \log\sigma(w\cdot x+b) + (1-y)\log(1-\sigma(w\cdot x+b))\right]$$

Cross-entropy loss

- Going back to the sentiment analysis example. We want loss to be:
  - smaller if the model estimate is close to correct
  - bigger if model is confused

- Let's first suppose the true label of this is y=1 (positive)

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

- What’s the cross-entropy loss for when w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] and b = 0.1?

Cross-entropy loss

- True value is y=1. How well is the model doing?

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
&= \sigma(.833) \\
&= 0.70 \qquad (5.6)
\end{aligned}
$$

- Pretty well. What's the loss?

$$
\begin{aligned}
L_{\mathrm{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1-y) \log(1 - \sigma(w \cdot x + b))] \\
&= -[\log \sigma(w \cdot x + b)] \\
&= -\log(.70) \\
&= .36
\end{aligned}
$$

Cross-entropy loss

- Suppose, for the same model, the true value is y=0 instead (i.e. it's a negative review). How well is the model doing?

$$\begin{aligned} p(-|x) = P(Y=0|x) &= 1-\sigma(w\cdot x+b) \\ &= 0.30 \end{aligned}$$

- Not so well this time. What's the loss?

$$\begin{aligned} L_{\text{CE}}(\hat{y},y) &= -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))] \\ &= -[\log(1-\sigma(w\cdot x+b))] \\ &= -\log(.30) \\ &= 1.2 \end{aligned}$$

- Loss is higher. More of a change is needed to the weights to predict this example correctly.

Cross-entropy loss

- For multinomial output, a **softmax cross-entropy loss** can be used
- The true output/label is a **one-hot vector**, where for a ground-truth label *i,* $y_i = 1$ and it's 0 everywhere else:

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -\sum_{k=1}^{K} \mathbb{1}\{y = k\} \log \hat{y}_i \\ &= -\sum_{k=1}^{K} \mathbb{1}\{y = k\} \log \hat{p}(y = k|x) \\ &= -\sum_{k=1}^{K} \mathbb{1}\{y = k\} \log \frac{\exp(\mathbf{z}_k)}{\sum_{j=1}^{K} \exp(\mathbf{z}_j)} \end{aligned}$$

- The terms in the sum will be 0, except for the term corresponding to the true class, so, this can be simplified to cross-entropy, also called the **negative log loss**.

$$L_{CE}(\hat{y}, y) = -\log \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{K} \exp(\mathbf{z}_j)}$$

Gradient descent

- We want to change/learn the weights to **minimise cross-entropy loss** over the training data.

- To do this, we want to find a **loss curve** and change weights such that the predictions get to the bottom of it.

- For logistic regression, loss function is **convex** (i.e. has just one minimum):
    - Gradient descent starting from any point is **guaranteed to find the minimum for logistic regression**
    - Note: loss for deep neural networks is non-convex.

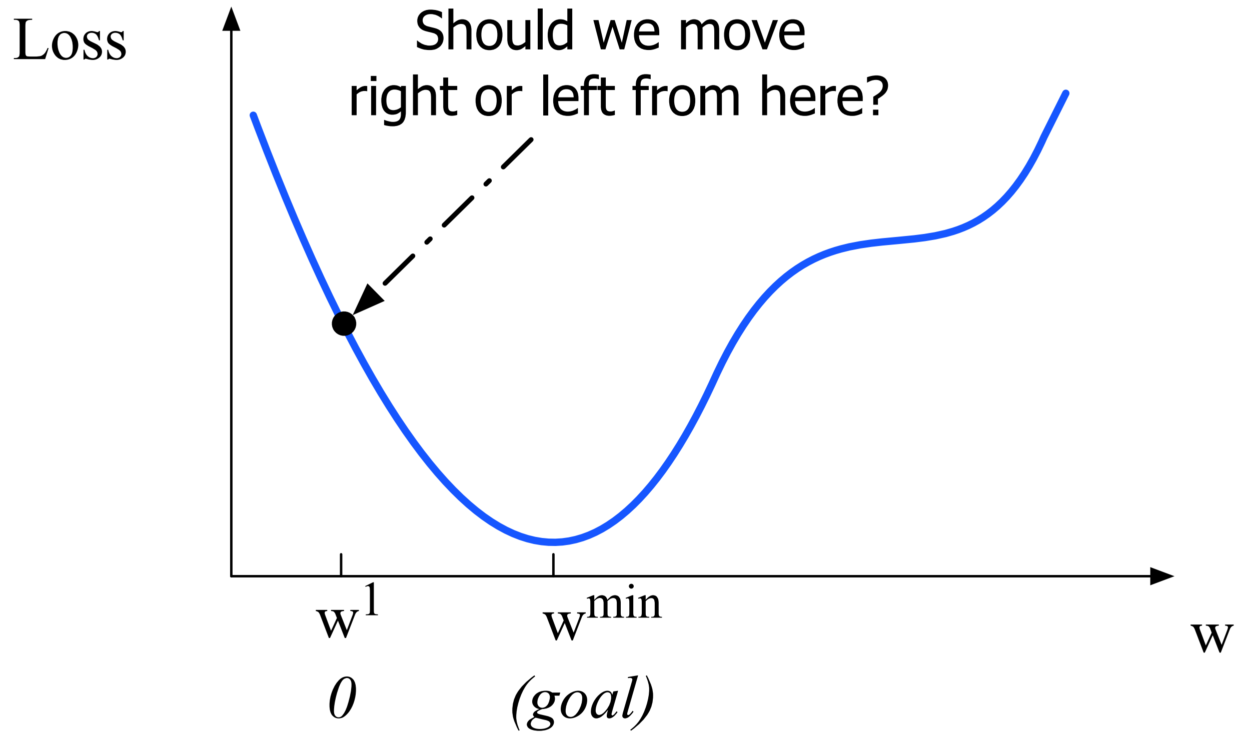
- (More detail in Neural Nets unit.)

Gradient descent

We use a method called **gradient descent.**

Q: Given current *w*, should we make it bigger or smaller?

A: Move *w* in the reverse direction from the slope of the function



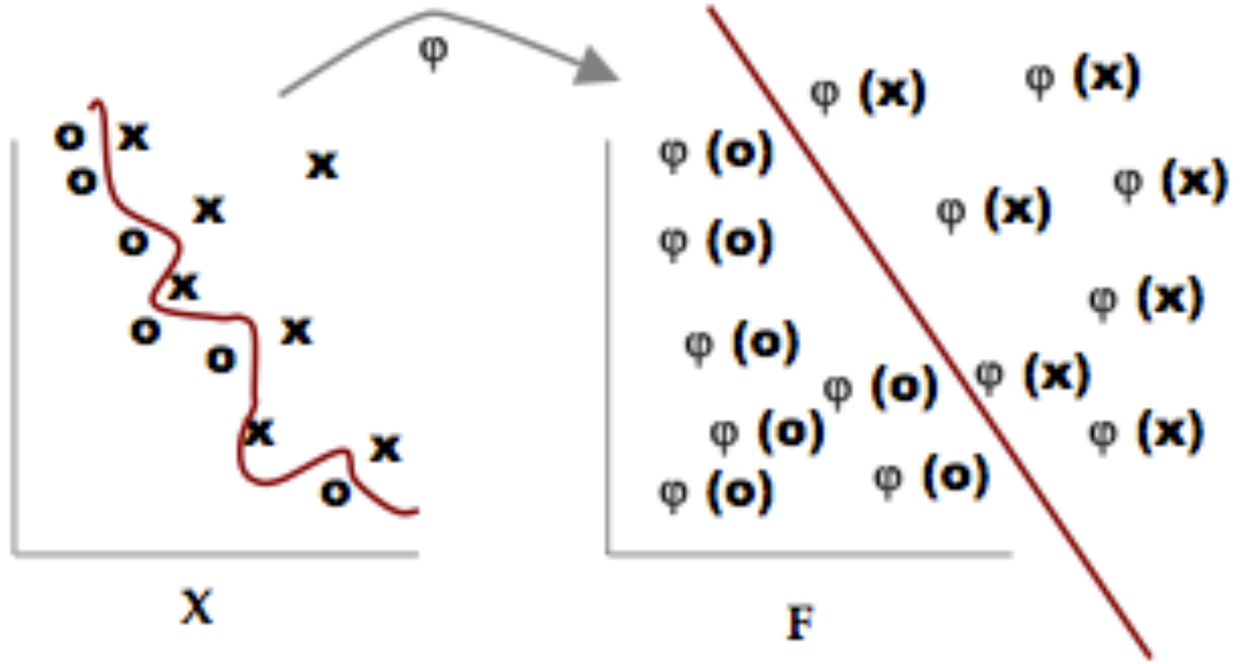
GD finds the gradient of the loss function at the current point and moves in the opposite direction (down the slope).

OUTLINE

1) General feature extraction functions for text classification
2) Feature selection and weighing methods
3) Going discriminative: Logistic Regression
4) Alternative linear classifier: SVMs

Linear Classifiers

- An alternative to using something like LR and the exponential function, is to keep to linear classification, but to **transform the dataset** into a dimension in which the classes are linearly separable.

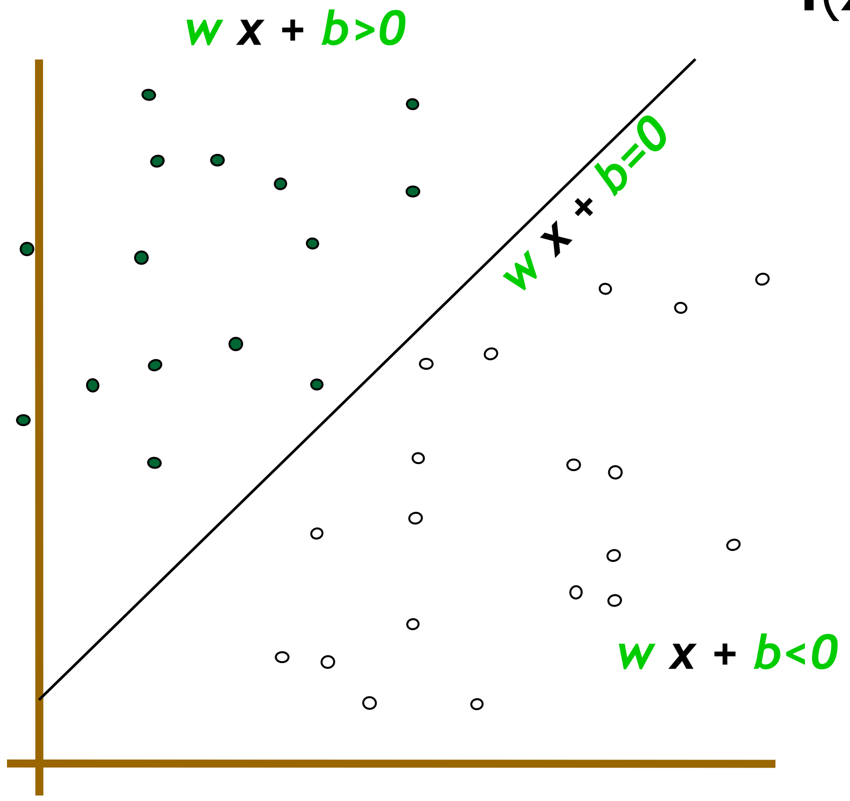


Linear Classifiers

● denotes +1

○ denotes -1

$\mathbf{f}(\mathbf{x}) = \mathbf{w}\,\mathbf{x} + b$

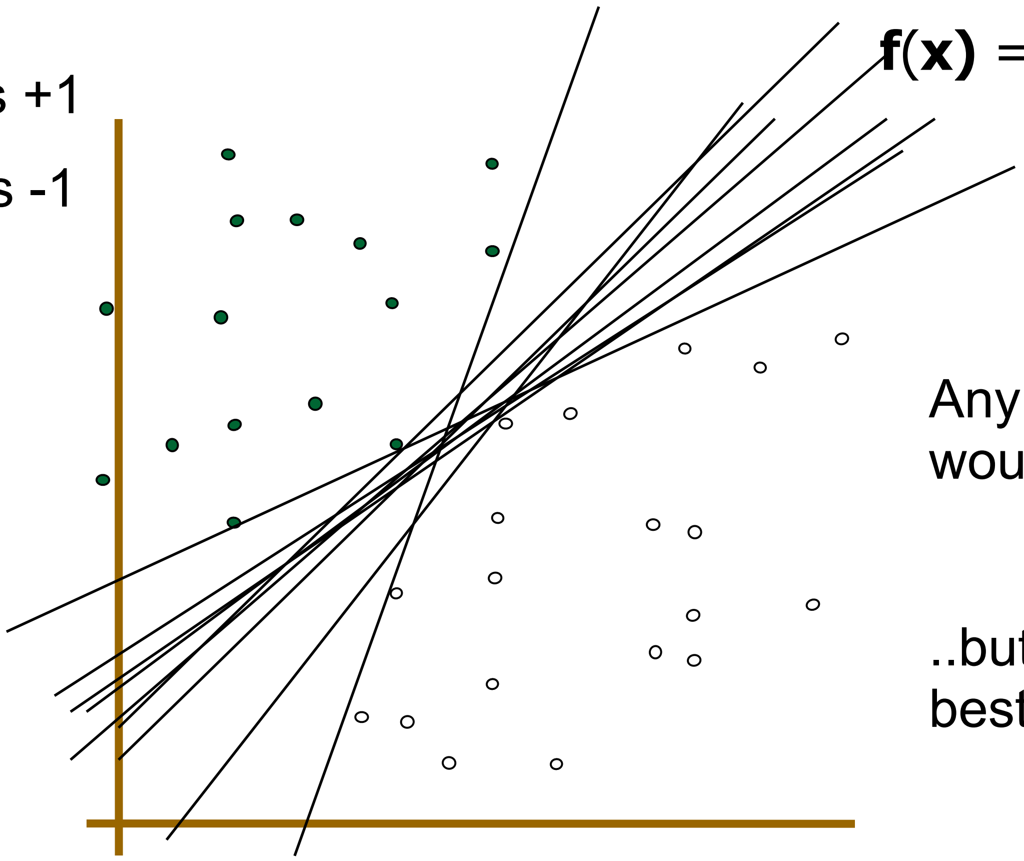


How would you classify this data?

Linear Classifiers

$$f(\mathbf{x}) = \mathbf{w}\ \mathbf{x} + b$$

- ● denotes +1
- ○ denotes -1



Any of these would be fine..

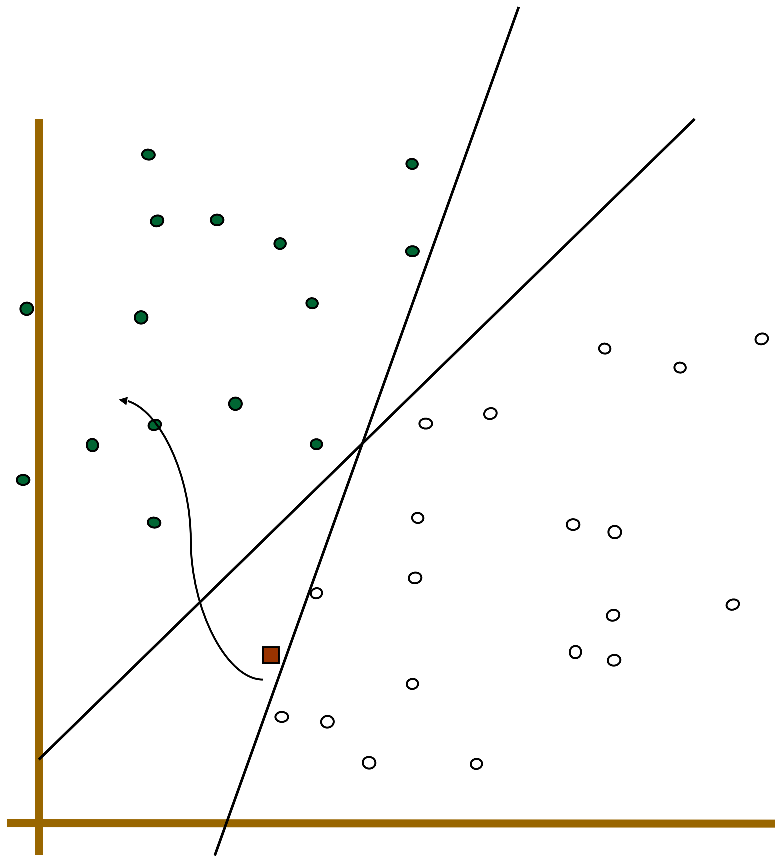
..but which is best?

Linear Classifiers

● denotes +1

○ denotes -1

$\mathbf{f}(\mathbf{x}) = \mathbf{w}\ \mathbf{x} + b$



How would you classify this data?

Misclassified to +1 class

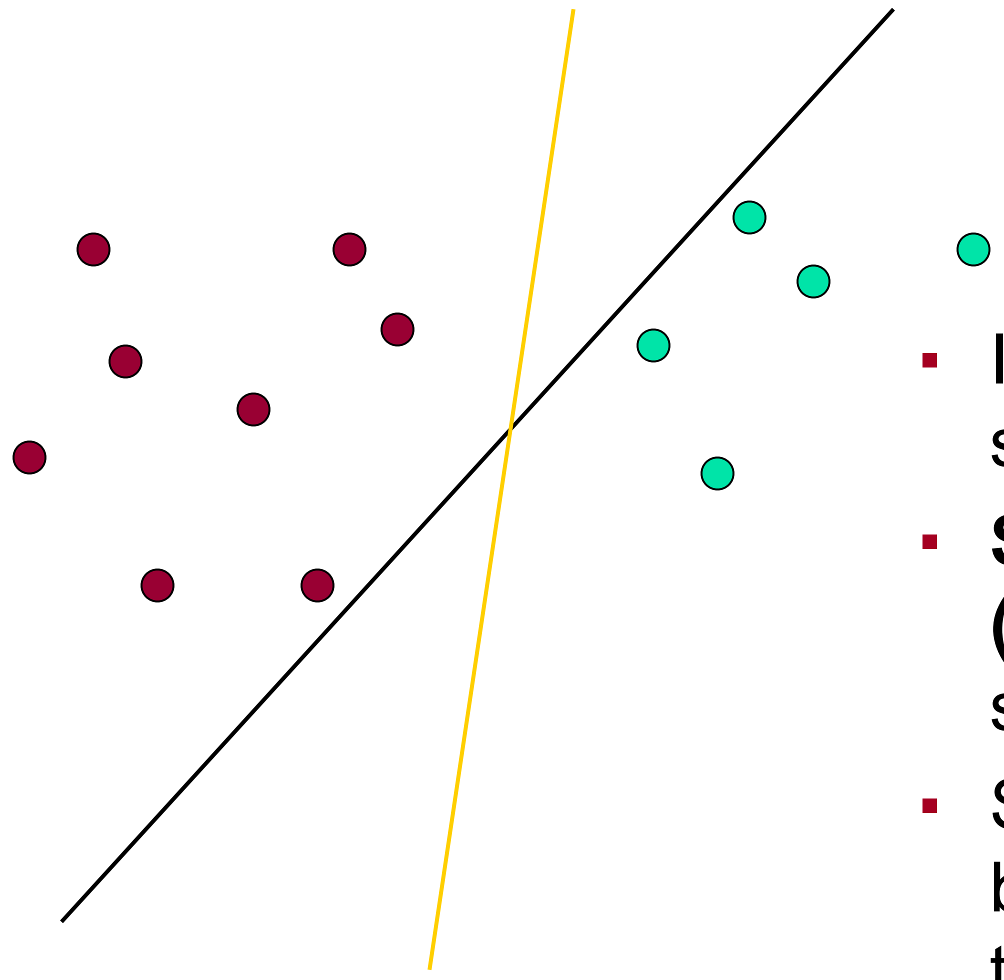
Linear Classifiers: summary

- Many common text classifiers are linear classifiers
- Despite this similarity, large performance differences
  - For separable problems, there is an infinite number of separating hyperplanes. Which one do you choose?
  - What to do for non-separable problems?

Separation by Hyperplanes

- Assume *linear separability* for now:
  - in 2 dimensions, can separate by a line
  - in higher dimensions, need hyperplanes
- Can find separating hyperplane by *linear programming* (e.g. perceptron):
  - separator can be expressed as $ax + by = c$

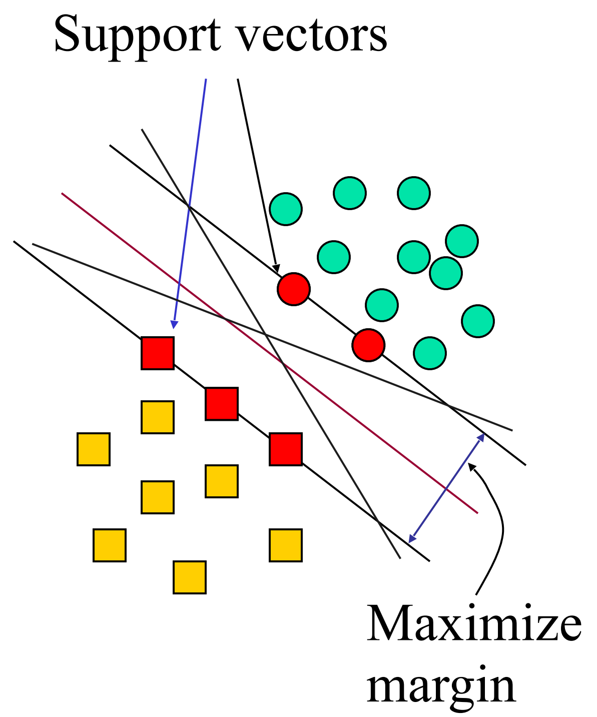
Which Hyperplane?



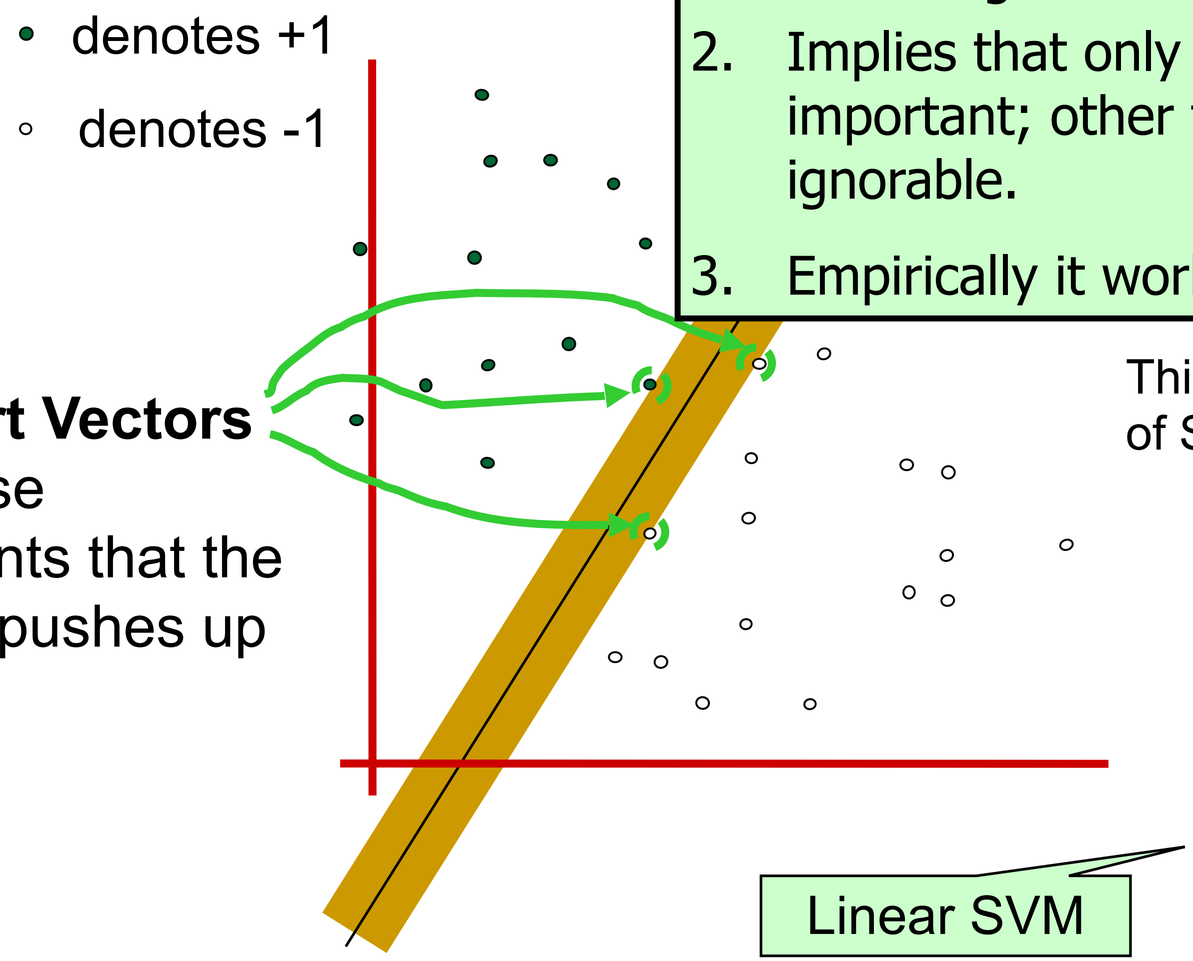
- In general, lots of possible solutions for *a,b,c*.
- **Support Vector Machine (SVM)** finds an optimal solution.
- SVMs answer: which line best splits the data into the two groups?

Support Vector Machine (SVM)

- SVMs **maximize the margin** around the separating hyperplane.
  - The distance between the points and the line are is as far as possible.
- The decision function is fully specified by a subset of training samples, *the support vectors*.
- **Constraint optimisation problem:**
  - Optimisation is maximise the margin
  - Constraint is that the data points can't be on the margins.



Maximum Margin



1. Maximizing the margin is good according to intuition and PAC theory
2. Implies that only support vectors are important; other training examples are ignorable.
3. Empirically it works very very well.

Support Vectors are those datapoints that the margin pushes up against

This is the simplest kind of SVM (Called an LSVM)

Linear SVM

Maximum Margin: Formulation

- w: hyperplane normal vector
- x_i: data point i
- y_i: class of data point i (+1 or -1)

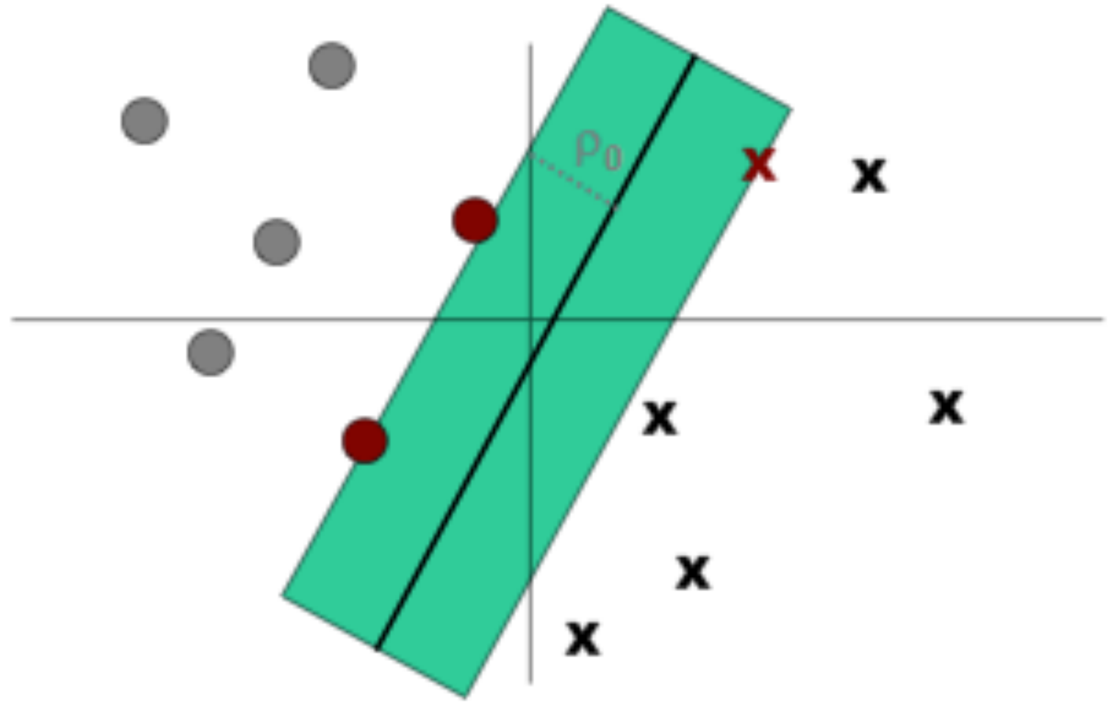
- Constraint optimization formalization:

- (1)
$$\begin{aligned} \mathbf{x}_i \cdot \mathbf{w} + b &\geq +1 \quad \text{for } y_i = +1 \\ \mathbf{x}_i \cdot \mathbf{w} + b &\leq -1 \quad \text{for } y_i = -1 \end{aligned}$$
- (2) maximize margin: 2/||w|| (i.e. distance of point from hyperplane)

Maximum Margin in geometrical terms

Support Vectors: Input vectors for which

$w_0^T x + b_0 = 1$ or $w_0^T x + b_0 = -1$



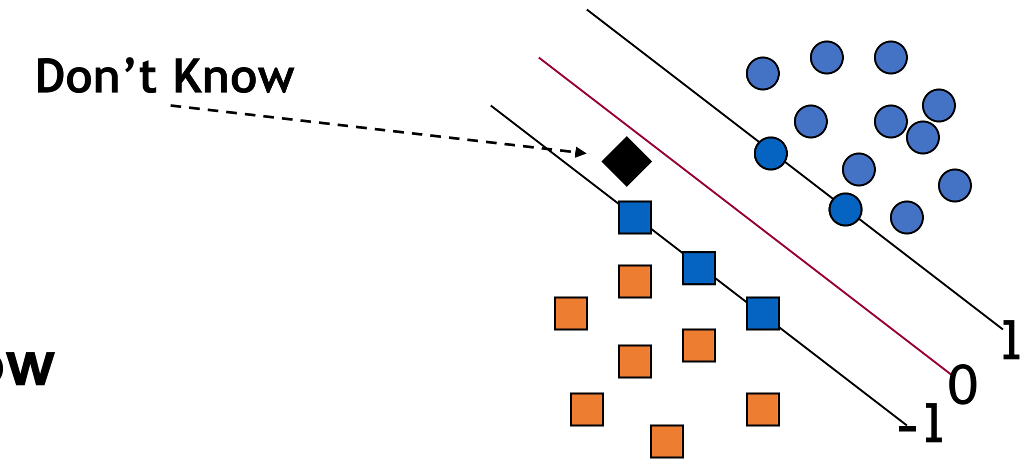
Classification with SVMs

- Given a new point **x**, we can score its projection onto the hyperplane normal:
  - I.e., compute score: $\mathbf{w}^T\mathbf{x} + b = \Sigma \alpha_i y_i \mathbf{x_i}^T\mathbf{x} + b$
  - Decide class based on whether < 0 or > 0
  - Can set confidence threshold *t*.

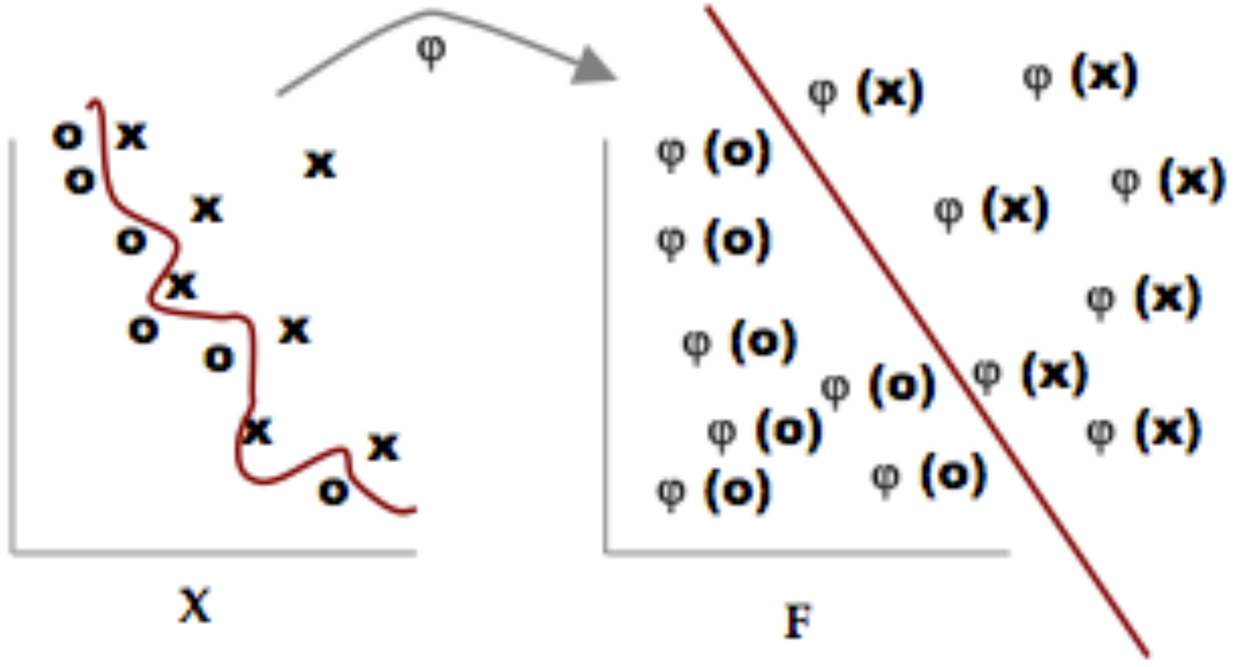
Score > t: **yes**

Score < -t: **no**

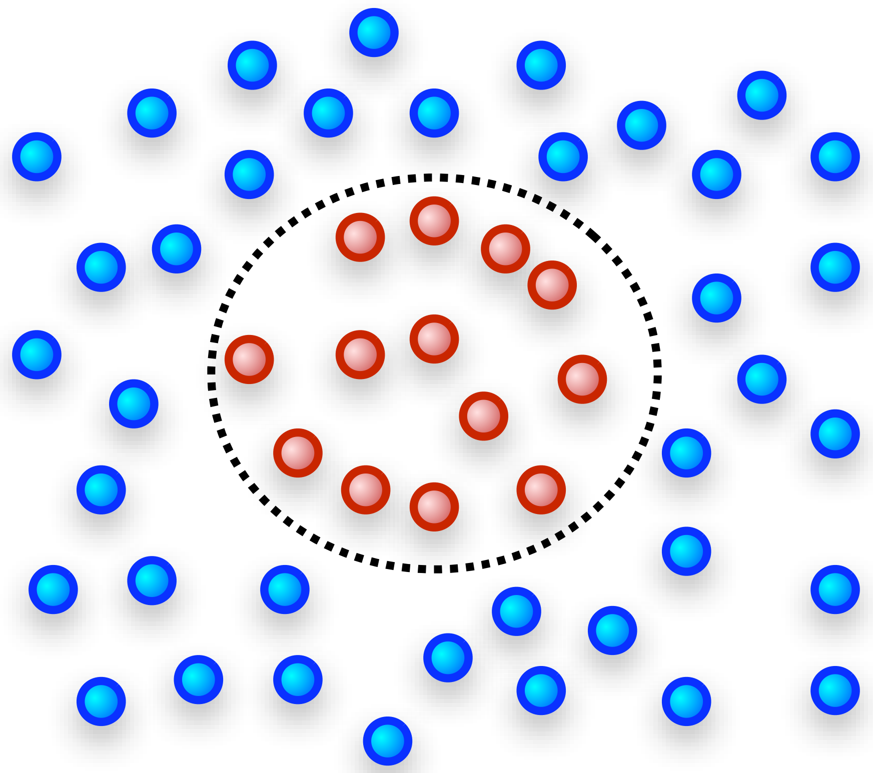
Else: **don't know**



Non-linear mapping can make a non-linear problem linearly separable



Non-linear mapping can make a non-linear problem linearly separable



Kernels

SVMs vs other approaches: Classic Reuters-21578 Data Set

- Most (over)used data set.
- 21578 documents:
  - 9603 training, 3299 test articles
- 118 categories.
  - An article can be in more than one category.
  - Learn 118 binary category distinctions.
- Average document: about 90 types, 200 tokens.
- Quite sparse problem: only about 10 out of 118 categories are large.

Common categories (#train, #test)

- Earn (2877, 1087)
- Acquisitions (1650, 179)
- Money-fx (538, 179)
- Grain (433, 149)
- Crude (389, 189)
- Trade (369,119)
- Interest (347, 131)
- Ship (197, 89)
- Wheat (212, 71)
- Corn (182, 56)

Reuters Text Categorization data set (Reuters-21578) document

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET"
OLDID="12981" NEWID="798">

<DATE> 2-MAR-1987 16:51:43.42</DATE>

<TOPICS><D>livestock</D><D>hog</D></TOPICS>

<TITLE>AMERICAN PORK CONGRESS KICKS OFF TOMORROW</TITLE>

<DATELINE>    CHICAGO, March 2 - </DATELINE><BODY>The American Pork Congress kicks
off tomorrow, March 3, in Indianapolis with 160 of the nations pork producers from 44 member states
determining industry positions on a number of issues, according to the National Pork Producers
Council, NPPC.

    Delegates to the three day Congress will be considering 26 resolutions concerning various issues,
including the future direction of farm policy and the tax law as it applies to the agriculture sector. The
delegates will also debate whether to endorse concepts of a national PRV (pseudorabies virus) control
and eradication program, the NPPC said.

    A large trade show, in conjunction with the congress, will feature the latest in technology in all
areas of the industry, the NPPC added. Reuter

&#3;</BODY></TEXT></REUTERS>
```

(a)

	NB	Rocchio	kNN	SVM
micro-avg-L (90 classes)	80	85	86	89
macro-avg (90 classes)	47	59	60	60

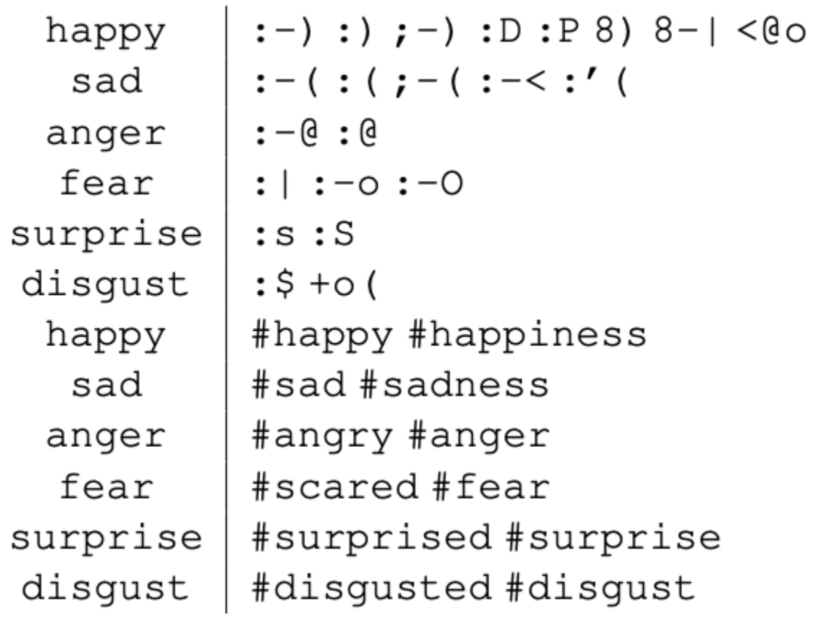
(b)

	NB	Rocchio	kNN	trees	SVM
earn	96	93	97	98	98
acq	88	65	92	90	94
money-fx	57	47	78	66	75
grain	79	68	82	85	95
crude	80	70	86	85	89
trade	64	65	77	73	76
interest	65	63	74	67	78
ship	85	49	79	74	86
wheat	70	69	77	93	92
corn	65	48	78	92	90
micro-avg (top 10)	82	65	82	88	92
micro-avg-D (118 classes)	75	62	n/a	n/a	87

Evaluation measure: F_1

Emotion classification in Twitter using distant supervision

- Purver and Battersby (2012)
  - Created dataset of Tweets and labelled them automatically according to markers from self-‘labelling’ by users, e.g. emoticons and hashtags.
  - These markers were used to generate the label then removed from the Tweet for training/testing.
  - Good use of automatic labelling without hand annotation - allows much more data!
  - Used unigram features and linear SVM classifier.
  - Performance: **F1=0.7+** for several hand-annotated emotions.



happy	:-) :) ;-) :D :P 8) 8-| <@o
sad	:-(:(;-(:-< :’(
anger	:-@ :@
fear	:| :-o :-O
surprise	:s :S
disgust	:$ +o(
happy	#happy #happiness
sad	#sad #sadness
anger	#angry #anger
fear	#scared #fear
surprise	#surprised #surprise
disgust	#disgusted #disgust

Detecting deceptive/fake Amazon Reviews

- Fornaciari and Poesio (2014)
  - Created dataset of Amazon reviews by collecting all reviews that were officially announced as being deceptive
  - Used unigrams, bigrams and trigrams as features
  - Selected using Information Gain
  - Using a SVM classifier
  - Performance: around **F1=0.77**

Summary: Discriminative vs. Generative

- SVMs and other discriminative classifiers can outperform NB and generative approaches with enough data.
- However, SVMs and other more complex approaches are less interpretable than Bayesian approaches.
- Logistic Regression can give a bit of the best of both worlds, as its co-efficients (feature weights) can be interpreted, and it can reach high-performance with optimization.

READING

- Jurafsky and Martin (2021, online)
  - Chapter 4 (Naïve Bayes and Classification Evaluation measures)
  - Chapter 5 (Logistic Regression)

- (Optional) Purver, M. and Battersby, S., 2012. Experimenting with distant supervision for emotion classification. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics* (pp. 482-491)
- (Optional) Fornaciari, T., & Poesio, M., 2014, Identifying fake Amazon reviews as learning from crowds. n *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics* (pp. 279–287)