Information Retrieval
(Text Categorization)

Fabio Aiolli
http://www.math.unipd.it/~aiolli

Dipartimento di Matematica Pura ed Applicata
Università di Padova

Anno Accademico 2008/2009

Text Categorization

- Text categorization (TC - aka text classification) is the task of buiding text classifiers, i.e. sofware systems that classify documents from a domain D into a given, fixed set $C = \{c_1,\ldots,c_m\}$ of categories (aka classes or labels)
- TC is an approximation task, in that we assume the existence of an 'oracle', a target function that specifies how docs ought to be classified.
- Since this oracle is unknown, the task consists in building a system that 'approximates' it

Text Categorization

- We will assume that categories are symbolic labels; in particular, the text constituting the label is not significant. No additional knowledge of category 'meaning' is available to help building the classifier

- The attribution of documents to categories should be realized on the basis of the content of the documents. Given that this is an inherently subjective notion, the membership of a document in a category cannot be determined with certainty

Single-label vs Multi-label TC

- TC comes in two different variants:
  - Single-label TC (SL) when exactly one category should be assigned to a document
    - The target function in the form $f : D \rightarrow C$ should be approximated by means of a classifier $f' : D \rightarrow C$
  - Multi-label TC (ML) when any number $\{0,\ldots,m\}$ of categories can be assigned to each document
    - The target function in the form $f : D \rightarrow P(C)$ should be approximated by means of a classifier $f' : D \rightarrow P(C)$
- We will often indicate a target function with the alternative notation $f : D \times C \rightarrow \{-1,+1\}$. Accordingly, a document d_j is called a positive example of c_i if $f(d_i,c_i)=+1$, and a negative example of c_i if $f(d_j,c_i)=-1$

Category and document pivoted categorization

- We may want to apply a ML classifier in to alternative ways:
  - Given $d_j \in D$, find all the category c_i under which it should be filed (document-pivoted categorization - DPC)
  - Given $c_i \in C$, find all the documents d_j that should be filed under it (category-pivoted categorization - CPC)
- The distinction may be important when the sets C or D are not available in their entirety from the start
  - DPC is suitable when documents become available one at a time (e.g. in e-mail filtering)
  - CPC is suitable when a new category c_{m+1} is added to C after a number of docs have already been classified under C (e.g. in patent classification)

"Hard" and "soft" categorization

- Fully automated classifiers need to take a 'hard' binary decision for each pair $\langle d_j, c_i \rangle$
- Semi-automated, 'interactive' classifiers are instead obtained by allowing 'soft' (i.e. real-valued) decisions:
  - Given $d_j \in D$ a system might rank the categories in C according to their estimated appropriateness to d_j (category ranking task)
  - Given $c_i \in C$ a system might rank the documents in D according to their estimated appropriateness to c_i (document ranking task)

"Hard" and "soft" categorization

- Such ranked lists would be of great help to a human expert in charge of taking the final categorization decision
  - They can thus restrict to the items at the top, rather than having to examine the entire set
- Semi-automated classifiers are useful especially in critical applications where the effectiveness of an automated system may be significantly lower than that of a human expert

Application of TC

TC has been used in a number of different applications

- Automatic indexing for Boolean IR
- Document organization
- Document filtering (e-mail filtering, spam filtering)
- Categorization of web pages into hierarchical catalogues

Automatic indexing for Boolean IR

- The application that spawned most of the early research in TC is that of automatic document indexing for use in Boolean IR systems
- Each document is assigned one or more keywords belonging to a controlled dictionary. Usually, this is performed by trained human indexers, thus a costly activity
- If the entries in the controlled dictionary are viewed as categories, document indexing is an instance of TC
- This is a multi-label task, and document-pivoted categorization is used
- This form of automated indexing may also be viewed as a form of automated metadata generation (or ontology learning), which is going to be very important for the 'Semantic Web'

Document Organization

- Many issues pertaining to document organization and filing, be it for purposes of personal organization or document repository structuring, may be addressed by automatic categorization techniques.
- Possible instances are:
  - Classifying 'incoming' articles at a newspaper
  - Grouping conference papers into sessions
  - Assigning a paper to a review to the more appropriate expert reviewer
  - Classifying patents for easing their later retrieval

Document filtering

- Document filtering (DF) is the categorization of a dynamic stream of incoming documents dispatched in an asynchronous way by an information consumer
- A typical example is a newsfeed (the information producer is a news agency and the information consumer is a newspaper). In this case, the DF system should discard the documents the consumer is not likely to be interested in
- A DF system may be installed
  - At the producer end - to route the info to the interested users only
  - At the consumer end – to block the delivery of info deemed interesting to the consumer

Other applications

- Author (or author's gender) identification for documents of disputed paternity [deVel01,Koppel02,Diederich03]
- Automatic identification of text genre [Finn02,Lee&Myaeng02,Liu03] or Web page genre [MeyerZuEissen&Stein04]
- Polarity detection (aka 'sentiment classification') [Pang02,Turmey02,Kim&Hovy04]
- Multimedia document categorization through caption analysis [Sable&Hatzivassiloglu99]
- Speech categorization through speech recognition + TC [Myers00,Schapire&Singer00]
- Automatic survey coding [Giorgetti&Sebastiani03]
- Text-to-speech synthesis for news reading [Alias02]
- Question type classification for question answering [Li&Roth02,Zhang&Lee03]

Machine Learning (ML) & TC

- In the '80s, the typical approach used for the construction of TC system involved hand-crafting an expert system consisting of a set of rules, one per category, of the form
  - If <DNF formula> then <category> else ¬<category>
  - Where <DNF formula> is a disjunction of conjunctive clauses
- The drawback of this "manual" approach is the knowledge acquisition bottleneck: since rules must be manually defined, building a classifier is expensive, and if the set of categories is updated or the classifier is ported to a different domain, other manual work has to be done.

Example

If	((wheat & farm)	**or**	
	(wheat & commodity)	**or**	
	(bushels & export)	**or**	
	(wheat & tonnes)	**or**	
	(wheat & winter & :soft))	**then**	WHEAT else : WHEAT

Induction

- Since the early '90s, the machine learning approach to the construction of TC systems has become dominant.
- A general inductive process automatically builds a classifier for a category c by 'observing' the characteristics of a set of documents previously classified belonging (or not) to c_i, by a domain expert.
- This is an instance of Supervised Learning (SL).

Advantages of the SL approach

- The engineering effort goes towards the construction, not of a classifier, but of an automatic builder of classifiers (learner)
- If the set of categories is updated, or if the system is ported to a different domain, all that is need is a different set of manually classified documents
- Domain expertise (for labeling), and not knowledge engineering expertise, is needed. Easier to characterize a concept extensionally than intentionally.
- Sometimes the preclassified documents are already available
- The effectiveness achievable nowadays by these classifiers rivals that of hand-crafted classifiers and that of human classifiers

Training Set and Test Set

- The ML approach relies on the application of a train-and-test approach to a labeled corpus $Tr=\{d_1,\ldots,d_{|S|}\}$, i.e. a set of documents previously classified under $C=\{c_1,\ldots,c_m\}$.
- The value of the function $\Phi : D \times C \rightarrow \{-1,+1\}$ are known for every pair $<d_j,c_i>$. Tr then constitutes a 'glimpse' of the ground truth.
- We assume that pair $<d_j,c_i>$ are extracted according to a probability distribution $P(d_j,c_i)$
- For evaluation purposes, a new set Te is usually provided which has elements extracted from the same pair distribution $P(d_j,c_i)$ used for elements in Tr.

Model Selection and Hold-out

- Most of the time, the learner is parametric. These parameters should be optimized by testing which values of the parameters yield the best effectiveness.
- Hold-out procedure
  1. A small subset of Tr, called the validation set (or hold-out set), denoted Va, is identified
  2. A classifier is learnt using examples in Tr-Va.
  3. Step 2 is performed with different values of the parameters, and tested against the hold-out sample
- In an operational setting, after parameter optimization, one typically re-trains the classifier on the entire training corpus, in order to boost effectiveness (debatable step!)
- It is possible to show that the evaluation performed in Step 2 gives an unbiased estimate of the error performed by a classifier learnt with the same parameters and with training set of cardinality $|Tr|-|Va|<|Tr|$

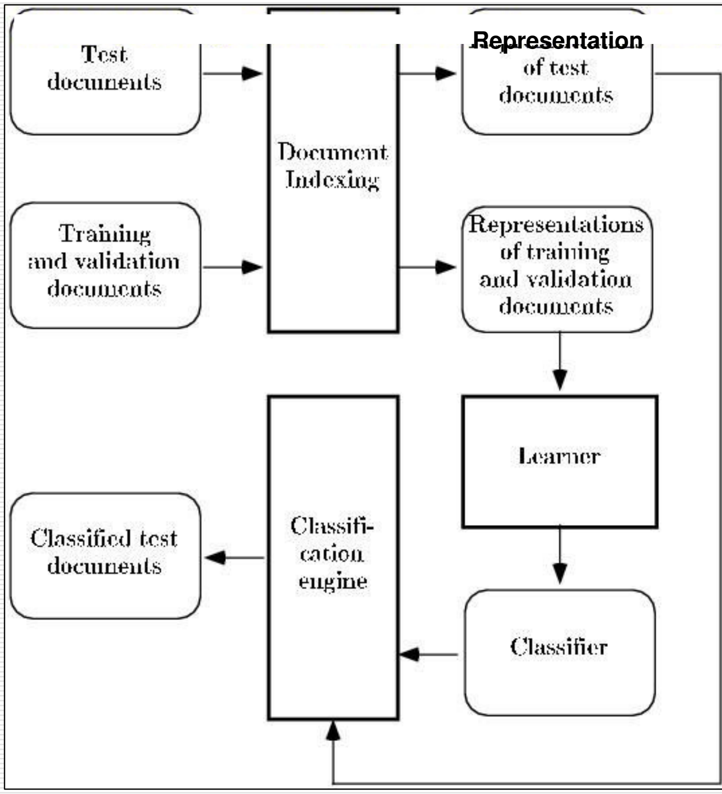
K-fold Cross Validation

- An alternative approach to model selection (and evaluation) is the K-fold cross-validation method
- K-fold CV procedure
  - K different classifiers $h_1, h_2, \ldots, h_k$ are built by partitioning the initial corpus Tr into k disjoint sets $Va_1, \ldots, Va_k$ and then iteratively applying the Hold-out approach on the k-pairs $\langle Tr_i = Tr - Va_i, Va_i \rangle$
  - Effectiveness is obtained by individually computing the effectiveness of $h_1, \ldots, h_k$, and then averaging the individual results
- The special case k=|Tr| of k-fold cross-validation is called leave-one-out cross-validation

The TC Process

The text categorization process consists of the following phases

- Document Indexing. This take as input the training, validation, and test documents, and outputs internal representations for them. Techniques of traditional IR (and information theory)
- Classifier Learning. This takes as input the representations of the training and validation sets and outputs a classifier. Techniques of ML.
- Classifier Evaluation. This takes as input the results of the classification of the test set, and is mostly accomplished by evaluation techniques belonging to both the IR and the ML tradition.



Architecture of a text classification system

Evaluation for Text Categorization

- Classification accuracy:
  - usual in ML,
  - the proportion of correct decisions,
  - Not appropriate if the population rate of the class is low
- Precision, Recall and F_1
  - Better measures

Evaluation for sets of classes

- How can we combine evaluation w.r.t. single classes into an evaluation for prediction over multiple classes?
- Two aggregate measures
  - Macro-Averaging, computes a simple average over the classes of the precision, recall, and F_1 measures
  - Micro-Averaging, pools per-doc decisions across classes and then compute precision, recall, and F_1 on the pooled contingency table

Macro and Micro Averaging

- Macro-averaging gives the same weight to each class
- Micro-averaging gives the same weight to each per-doc decision

Example

Class 1	Truth: "yes"	Truth: "no"
Pred: "yes"	10	10
Pred: "no"	10	970

Class 2	Truth: "yes"	Truth: "no"
Pred: "yes"	90	10
Pred: "no"	10	890

POOLED	Truth: "yes"	Truth: "no"
Pred: "yes"	100	20
Pred: "no"	20	1860

Macro-Averaged Precision: (.5+.9)/2 = .7

Micro-averaged Precision: 100/120 = .833…

Benchmark Collections (used in Text Categorization)

Reuters-21578

- The most widely used in text categorization. It consists of newswire articles which are labeled with some number of topical classifications (zero or more out of 115 classes). 9603 train + 3299 test documents

Reuters RCV1

- Newstories, larger than the previous (about 810K documents) and a hierarchically structured set of (103) leaf classes

Oshumed

- a ML set of 348K docs classified under a hierarchically structured set of 14K classes (MESH thesaurus). Title+abstracts of scientific medical papers.

20 Newsgroups

- 18491 articles from the 20 Usenet newsgroups

The inductive construction of classifiers

Two different phases to build a classifier h_i for category $c_i \in C$

1. Defintion of a function $CSV_i : D \rightarrow \mathcal{R}$, a categorization status value, representing the strength of the evidence that a given document d_j belongs to c_i
2. Definition of a threshold τ_i such that
   - $CSV_i(d_j) \geq \tau_i$ interpreted as a decision to classify d_j under c_i
   - $CSV_i(d_j) \leq \tau_i$ interpreted as a decision not to classify d_j under c_i

CSV and Proportional thresholding

- Two different ways to determine the thresholds τ_i once given CSV_i are [Yang01]
    1. CSV thresholding: τ_i is a value returned by the CSV_i function. May or may not be equal for all the categories. Obtained on a validation set
    2. Proportional thresholding: τ_i are the values such that the validation set frequencies for each class is as close as possible to the same frequencies in the training set
- CSV thresholding is theoretically better motivated, and generally produces superior effectiveness, but computationally more expansive
- Thresholding is needed only for 'hard' classification. In 'soft' classification the decision is taken by the expert, and the CSV_i scores can be used for ranking purposes

Probabilistic Classifiers

- Probabilistic classifiers view $CSV_j(d_i)$ in terms of $P(c_j|d_i)$, and compute it by means of the Bayes' theorem
    - $P(c_j|d_i) = P(d_i|c_j)P(c_j)/P(d_i)$
    - Maximum a posteriori Hypothesys (MAP) argmax $P(c_j|d_i)$
- Classes are viewed as generators of documents
- The prior probability $P(c_j)$ is the probability that a document d is in c_j

Naive Bayes Classifiers

Task: Classify a new instance D based on a tuple of attribute values $D = \langle x_1, x_2, \ldots, x_n \rangle$ into one of the classes $c_j \in C$

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

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

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

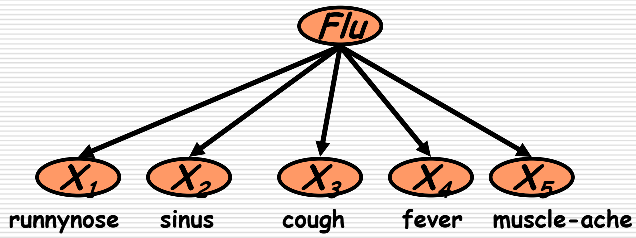
Naïve Bayes Classifier: Assumption

- $P(c_j)$
  - Can be estimated from the frequency of classes in the training examples.
- $P(x_1, x_2, \ldots, x_n | c_j)$
  - $O(|X|^n \cdot |C|)$ parameters
  - Could only be estimated if a very, very large number of training examples was available.

Naïve Bayes Conditional Independence Assumption:

- Assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities $P(x_i | c_j)$.

The Naïve Bayes Classifier

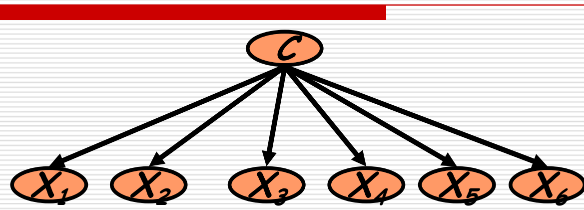


- **Conditional Independence Assumption:** features are independent of each other given the class:

$$P(X_1,\ldots,X_5 \mid C) = P(X_1 \mid C) \bullet P(X_2 \mid C) \bullet \cdots \bullet P(X_5 \mid C)$$

- Only n|C| parameters (+|C|) to estimate

Learning the Model

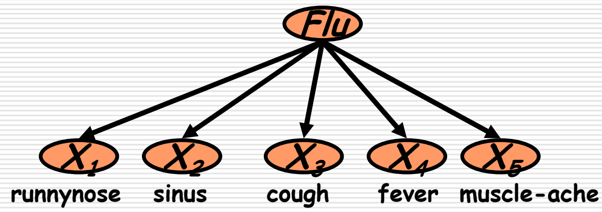


maximum likelihood estimates: most likely value of each parameter given the training data

- i.e. simply use the frequencies in the data

$$\hat{P}(c_j) = \frac{N(C = c_j)}{N} \qquad \hat{P}(x_i \mid c_j) = \frac{N(X_i = x_i, C = c_j)}{N(C = c_j)}$$

Problem with Max Likelihood



$$P(X_1,\dots,X_5 \mid C) = P(X_1 \mid C) \bullet P(X_2 \mid C) \bullet \cdots \bullet P(X_5 \mid C)$$

- What if we have seen no training cases where patient had no flu and muscle aches?

$$\hat{P}(X_5 = t \mid C = nf) = \frac{N(X_5 = t, C = nf)}{N(C = nf)} = 0$$

- Zero probabilities cannot be conditioned away, no matter the other evidence!

$$\ell = \arg\max_c \hat{P}(c) \prod_i \hat{P}(x_i \mid c)$$

Smoothing to Avoid Overfitting

$$\hat{P}(x_i \mid c_j) = \frac{N(X_i = x_i, C = c_j) + 1}{N(C = c_j) + k}$$

# of values of X_i

Stochastic Language Models

- Models *probability* of generating strings (each word in turn) in the language.

Model M

0.2	the
0.1	a
0.01	man
0.01	woman
0.03	said
0.02	likes
…	

the	man	likes	the	woman
0.2	0.01	0.02	0.2	0.01

multiply

P(s | M) = 0.00000008

Stochastic Language Models

- Model *probability* of generating any string

Model M1

0.2	the
0.01	class
0.0001	sayst
0.0001	pleaseth
0.0001	yon
0.0005	maiden
0.01	woman

Model M2

0.2	the
0.0001	class
0.03	sayst
0.02	pleaseth
0.1	yon
0.01	maiden
0.0001	woman

the	class	pleaseth	yon	maiden
0.2	0.01	0.0001	0.0001	0.0005
0.2	0.0001	0.02	0.1	0.01

P(s|M2) > P(s|M1)

Two Models

- Model 1: Multivariate binomial
  - One feature X_w for each word in dictionary
  - X_w = true in document d if w appears in d
  - Naive Bayes assumption:
    - Given the document's topic, appearance of one word in the document tells us nothing about chances that another word appears
- This is the model you get from binary independence model in probabilistic relevance feedback in hand-classified data

Two Models

- Model 2: Multinomial
  - One feature X_i for each word pos in document
    - feature's values are all words in dictionary
  - Value of X_i is the word in position i
  - Naïve Bayes assumption:
    - Given the document's topic, word in one position in the document tells us nothing about words in other positions
  - Second assumption:
    - Word appearance does not depend on position

$$P(X_i = w \mid c) = P(X_j = w \mid c)$$

Parameter estimation

- Binomial model:

$$\hat{P}(X_w = t \mid c_j) =$$ fraction of documents of topic c_j in which word w appears

- Multinomial model:

$$\hat{P}(X_i = w \mid c_j) =$$ fraction of times in which word w appears across all documents of topic c_j

Naïve Bayes: Learning

- From training corpus, extract *Vocabulary*
- Calculate required $P(c_j)$ and $P(x_k \mid c_j)$ terms
  - For each c_j in C do
    - $docs_j \leftarrow$ subset of documents for which the target class is c_j
    - $P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$
    - $Text_j \leftarrow$ single document containing all $docs_j$
    - for each word x_k in *Vocabulary*
      - $n_k \leftarrow$ number of occurrences of x_k in $Text_j$
      - $P(x_k \mid c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha |Vocabulary|}$

Naïve Bayes: Classifying

- positions ← all word positions in current document which contain tokens found in *Vocabulary*
- Return c_{NB}, where

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

Naive Bayes: Time Complexity

- Training Time: $O(|D|L_d + |C||V|)$
  - where L_d is the average length of a document in D.
  - Assumes V and all D_i, n_i, and n_{ij} pre-computed in $O(|D|L_d)$ time during one pass through all of the data.
  - Generally just $O(|D|L_d)$ since usually $|C||V| < |D|L_d$
- Test Time: $O(|C|\, L_t)$
  - where L_t is the average length of a test document.
- Very efficient overall, linearly proportional to the time needed to just read in all the data.

Underflow Prevention

- Multiplying lots of probabilities, which are between 0 and 1 by definition, can result in floating-point underflow.
- Since log(*xy*) = log(*x*) + log(*y*), it is better to perform all computations by summing logs of probabilities rather than multiplying probabilities.
- Class with highest final un-normalized log probability score is still the most probable.

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

Regression Models

- Regression means the approximation of a real-valued function $f : D \times C \rightarrow [-1,+1]$ by means of a function $h : D \times C \rightarrow [-1,+1]$ given a set of points d_i and their corresponding $f(d_i)$, the training data
- Various TC systems used regression models
- We describe the Linear Least Squares Fit (LLSF) approach of [Yang&Chute94]

Linear Least Squares Fit

- Classification can be seen as the task of attributing to test documents d_j an output vector O_j=<c_{1j},..,c_{mj}>, given its input vector I_j=<w_{1j},..,w_{nj}>
- Hence, building a classifier comes down to computing a $n \times m$ matrix M such that $I_j M = O_j$

Linear Least Squares Fit

- LLSF computes M from the training data by computing a linear least-squares fit that minimizes the error on the training set according to the formula

$$M = \arg\min_M ||IM - O||_F$$

  where
  - $||A||_F = \sqrt{\sum_{i=1}^{m} \sum_{i=1}^{n} a_{ij}^2}$ is the Frobenius norm of the matrix A
  - I is the |Tr| $\times$ n matrix of input vectors of the training docs
  - O is the |Tr| $\times$ m matrix of output vectors of the training docs
- The M matrix can be built by performing a singular value decomposition
- LLSF has been shown in [Yang99, Yang&Liu99] to be one of the most effective classifier. Its drawback is the high computational cost involved in computing M

LLSF by SVD

- The theory: you want to find a vector v such that Av is as close as possible to a vector b, i.e. to minimize the euclidean norm of the residuals ||Av-b||
- The derivative of $(Av-b)^t (Av-b)$ is $2A^tAv-2A^tb$
- Then the solution is at $A^tA\ v = A^t\ v$, i.e. $v = (A^tA)^{-1}A^t\ b$
- Let $A = USV^t$ the singular value decomposition of A and S is a diagonal matrix
- Then the pseudoinverse $(A^tA)^{-1}A^t = VS^+U^t$ and S^+ is S where each non zero entry is substituted by its reciprocal

Neural Networks

- A Neural Network (NN) TC system is a network of units:
  - Input units represent terms appearing in the document
  - Output units represent categories to be assigned
  - Hidden units are detectors that 'discover' correlations among terms present in the input
  - Adjustable weights are associated to connections between units
- NN are trained by the backpropagation algorithm: the activation of each pattern is propagated through the network, and the error produced is back propagated and the parameter changed to reduce the error
- Non linear NN components (hidden and output units) provide no advantage
- The use of NN for TC is declined in recent years

Decision Tree Classifiers

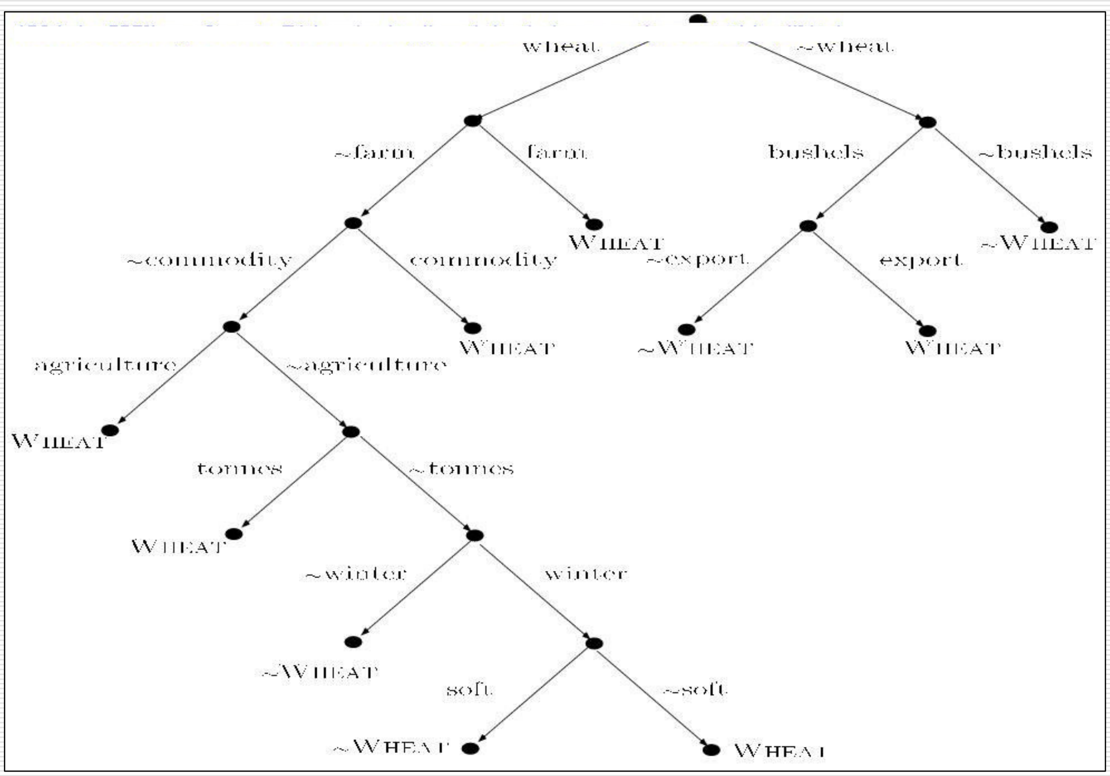
- A decision tree (DT) binary classifier for c_i consists of a tree in which
  - Internal nodes are labeled with terms
  - Branches departing from them correspond to the value that the term t_k has in the representation of test document d_j
  - Leaf nodes are labeled by categories
- The DT classifies d_j by recursively testing for the values of the term labeling, the internal nodes have in representation of d_k=j
- In binary TC, leaves are labeled by either c_i, or $\neg c_i$ while in SL TC they are labeled by one among the classes in C
- When the term weights w_{kj} are in {0,1}, DT are binary trees; when they are in [0,1], DTs have variable ariety, and the branches are labeled by intervals $[i_s, i_{s+1})$, where each interval results from an entropy based quantization of the [0,1] interval

Learning a binary DT

- A general procedure to build a binary DT for a category c_i consists in a divide and conquer strategy:
  1. Assign all the docs in Tr to the root, and make it the current node
  2. If the docs in the current node have all the same value for c_i, stop and label the node with this value; otherwise select the term t_k which maximally 'discriminates' between c_i and $\neg c_i$
  3. Partition the set of docs in the current node into sets of docs that have the same value for t_k
  4. Create a child node for each such possible value and recursively repeat the process from Step 2
- In Step 2, selecting the term t_k that maximally discriminates among the classes (by e.g. information gain (C4.5) or Gini index (CART)) tends to maximize the homogeneity (in terms of attached labels) of the sets produced in Step 3, and hence to minimize the depth of the tree

Learning a binary DT

- In order to avoid overfitting, the growth of the tree may be interrupted before excessively specific branches are produced
- Nowadays DT text classifiers are usually outperformed by more sophisticated learning methods



Entropy and Information Gain

- The entropy (very common in Information Theory) characterizes the (im)purity of an arbitrary collection of examples

- Information Gain is the expected reduction in entropy caused by partitioning the examples according to a given attribute

Entropy Calculations

If we have a set with k different values in it, we can calculate the entropy as follows:

$$entropy(Set) = I(Set) = -\sum_{i=1}^{k} P(value_i) \cdot \log_2\left(P(value_i)\right)$$

Where $P(value_i)$ is the probability of getting the i^{th} value when randomly selecting one from the set.

So, for the set R = {a,a,a,b,b,b,b,b}

$$entropy(R) = I(R) = -\left[\underbrace{\left(\frac{3}{8}\right)\log_2\left(\frac{3}{8}\right)}_{\text{a-values}} + \underbrace{\left(\frac{5}{8}\right)\log_2\left(\frac{5}{8}\right)}_{\text{b-values}}\right]$$

Looking at some data

Color	Size	Shape	Edible?
Yellow	Small	Round	+
Yellow	Small	Round	-
Green	Small	Irregular	+
Green	Large	Irregular	-
Yellow	Large	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Green	Small	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	+
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Small	Irregular	+
Yellow	Large	Irregular	+

Entropy for our data set

- 16 instances: 9 positive, 7 negative.

$$I(all\_data) = -\left[\left(\frac{9}{16}\right)\log_2\left(\frac{9}{16}\right) + \left(\frac{7}{16}\right)\log_2\left(\frac{7}{16}\right)\right]$$

- This equals: 0.9836
- This makes sense – it's almost a 50/50 split; so, the entropy should be close to 1.

Visualizing Information Gain

Size

Small | Large

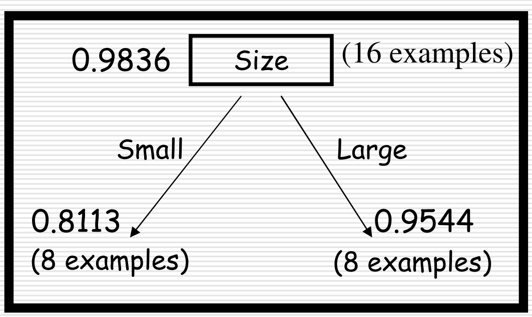
Color	Size	Shape	Edible?
Yellow	Small	Round	+
Yellow	Small	Round	-
Green	Small	Irregular	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Green	Small	Round	-
Yellow	Small	Irregular	+

Color	Size	Shape	Edible?
Green	Large	Irregular	-
Yellow	Large	Round	+
Yellow	Large	Round	-
Yellow	Large	Round	+
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Large	Irregular	+

$$G(S,A) = I(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} I(S_v)$$

Visualizing Information Gain

The data set that goes down each branch of the tree has its own entropy value. We can calculate for each possible attribute its **expected entropy**. This is the degree to which the entropy would change if branch on this attribute. You **add** the entropies of the two children, **weighted** by the proportion of examples from the parent node that ended up at that child.



Entropy of left child is 0.8113
I(size=small) = 0.8113

Entropy of right child is 0.9544
I(size=large) = 0.9544

$I(S_{Size})$ = (8/16)*.8113 + (8/16)*.9544 = .8828

G(attrib) = I(parent) – I(attrib)

We want to calculate the *information gain* (or entropy reduction). This is the reduction in 'uncertainty' when choosing our first branch as 'size'. We will represent information gain as "G."

$$G(\text{size}) = I(S) - I(S_{Size})$$
$$G(\text{size}) = 0.9836 - 0.8828$$
$$G(\text{size}) = 0.1008$$

Entropy of all data at parent node = I(parent) = 0.9836
Child's expected entropy for '**size**' split = I(size) = 0.8828

So, we have gained 0.1008 *bits* of information about the dataset by choosing 'size' as the first branch of our decision tree.

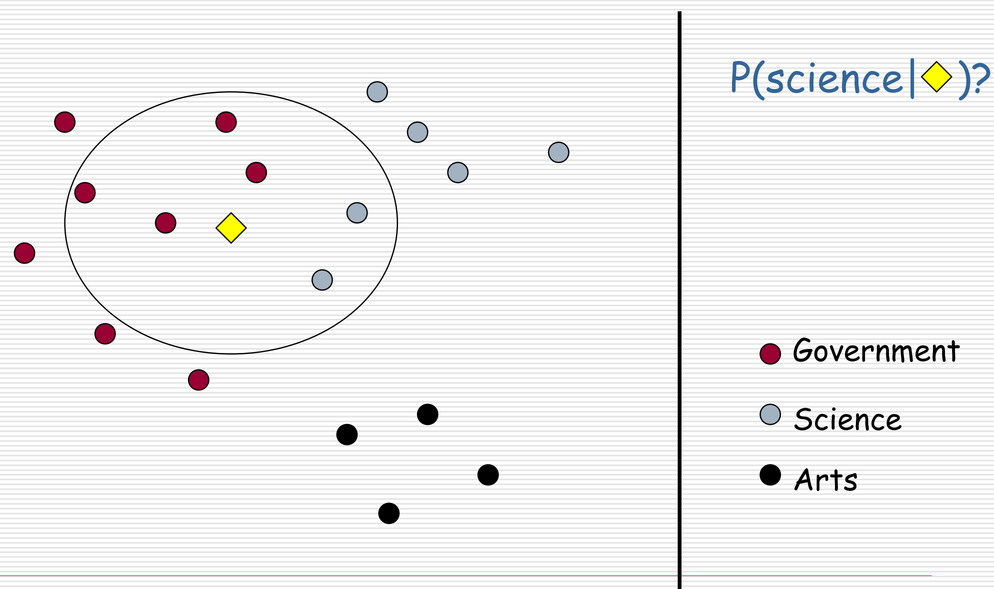
Example-based Classifiers

- Example-based classifiers (EBCs) learns from the categories of the training documents similar to the one to be classified
- The most frequently used EBC is the k-NN algorithm

k Nearest Neighbor Classification

1. To classify document *d* into class c
2. Define *k*-neighborhood N as *k* nearest neighbors (according to a given distance or similarity measure) of *d*
3. Count number of documents k_c in N that belong to c
4. Estimate P(c|*d*) as k_c/k
5. Choose as class argmax_c P(c|*d*)   [= majority class]

Example: k=6 (6NN)



Nearest-Neighbor Learning Algorithm

- Learning is just storing the representations of the training examples in *D*.
- Testing instance *x*:
  - Compute similarity between *x* and all examples in *D*.
  - Assign *x* the category of the most similar example in *D*.
- Does not explicitly compute a generalization or category prototypes.
- Also called:
  - Case-based learning
  - Memory-based learning
  - Lazy learning

kNN Is Close to Optimal

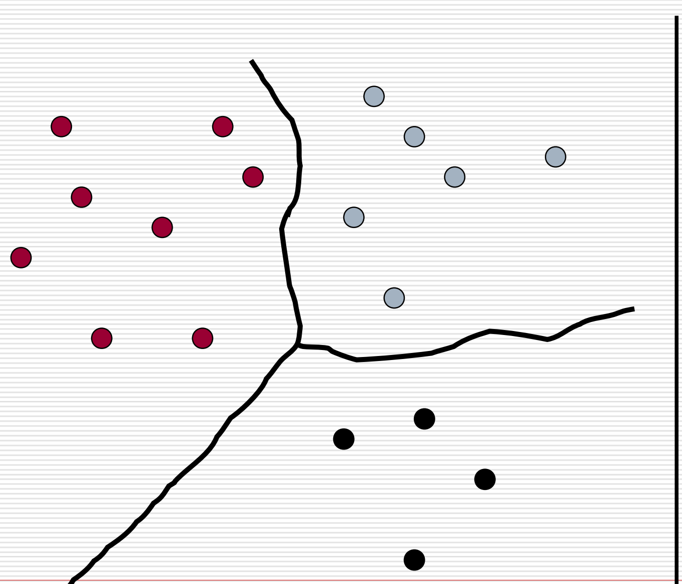
- Asymptotically, the error rate of 1-nearest-neighbor classification is less than twice the Bayes error [error rate of classifier knowing the model that generated data]

- Asymptotic error rate is 0 whenever Bayes error is 0.

K Nearest-Neighbor

- Using only the closest example to determine the categorization is subject to errors due to:
  - A single atypical example.
  - Noise (i.e. error) in the category label of a single training example.
- More robust alternative is to find the *k* most-similar examples and return the majority category of these *k* examples.
- Value of *k* is typically odd to avoid ties;

kNN decision boundaries



Boundaries are in principle arbitrary surfaces – but usually polyhedra

- Government
- Science
- Arts

Similarity Metrics

- Nearest neighbor method depends on a similarity (or distance) metric.
- Simplest for continuous *m*-dimensional instance space is *Euclidian distance*.
- Simplest for *m*-dimensional binary instance space is *Hamming distance* (number of feature values that differ).
- For text, cosine similarity of tf.idf weighted vectors is typically most effective.
- .. But any metric can be used!!

Nearest Neighbor with Inverted Index

- Naively finding nearest neighbors requires a linear search through $|D|$ documents in collection
- But determining *k* nearest neighbors is the same as determining the *k* best retrievals using the test document as a query to a database of training documents.
- Use standard vector space inverted index methods to find the *k* nearest neighbors.
- Testing Time: $O(B/V_t/)$ where *B* is the average number of training documents in which a test-document word appears.
  - Typically $B \ll |D|$

kNN: Discussion

- No feature selection necessary
- Scales well with large number of classes
  - Don't need to train *n* classifiers for *n* classes
- Classes can influence each other
  - Small changes to one class can have ripple effect
- Scores can be hard to convert to probabilities
- No training necessary

The Rocchio Method

- The Rocchio Method is an adaptation to TC of Rocchio's formula for relevance feedback. It computes a profile for c_i by means of the formula

$$w_{ki} = \frac{1}{|POS|} \sum_{d_j \in POS} w_{kj} - \delta \frac{1}{|NEG|} \sum_{d_j \in NEG} w_{kj}$$

  where POS = $\{d_j \in Tr | y_j=+1\}$ NEG = $\{d_j \in Tr | y_j=-1\}$, and δ may be seen as the ratio between γ and β parameters in RF

- In general, Rocchio rewards the similarity of a test document to the centroid of the positive training examples, and its dissimilarity from the centroid of the negative training examples
- Typical choices of the control parameter δ are $0 \leq \delta \leq .25$

Rocchio: a simple case study

When δ=0

- For each category (possibly, more than one), compute a *prototype* vector by summing the vectors of the training documents in the category.
  - Prototype = centroid of members of class
- Assign test documents to the category with the closest prototype vector based on cosine similarity.

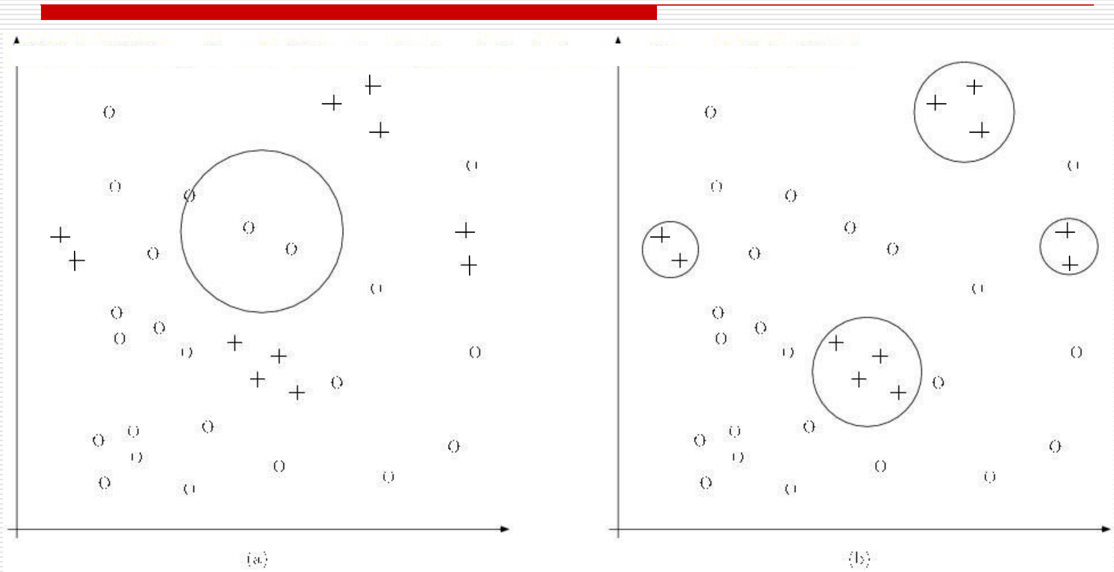
Rocchio Time Complexity

- Note: The time to add two sparse vectors is proportional to minimum number of non-zero entries in the two vectors.
- Training Time: $O(|D|L_d + |C||V|)) = O(|D| L_d)$ where L_d is the average length of a document in D and $|V|$ is the vocabulary size.
- Test Time: $O(L_t + |C||V_t|)$ where L_t is the average length of a test document and $|V_t|$ is the average vocabulary size for a test document.
  - Assumes lengths of **centroid** vectors are computed and stored during training, allowing cosSim(**d**, **c**$_i$) to be computed in time proportional to the number of non-zero entries in **d** (i.e. $|V_t|$)

The Rocchio Method (Pro and Cons)

- Advantages
  - The method is efficient and easy to implement
  - The resulting classifiers are easy interpretable. This does not happen for other approaches such as e.g. NNs
- Drawbacks
  - The resulting classifier are seldom very effective [Cohen&Singer96, Joachims98,Lewis+96,Yang99]
  - If the documents in c_j occur in disjoint clusters a Rocchio classifier may be very ineffective: as all linear classifers, it partition the space of documents in two linearly separable subspaces. But many naturally occurring categories are not linearly separable

Rocchio Vs k-NN



The Rocchio Method (Enhancements)

- Instead of considering the set of negative training instances in its entirely, a set of near-positives might be selected (as in RF). This is called the query zoning method

- Near positives are more significant, since they are the most difficult to tell apart from the positives. They may be identified by issuing a Rocchio query consisting of the centroid of the positive training examples against a document base consisting of the negative training examples. The top-ranked ones can be used as near positives.

- Some claim that, by using query zones plus other enhancements, the Rocchio method can achieve levels of effectiveness comparable to state-of-the art methods while being quicker to train

Linear Classifiers

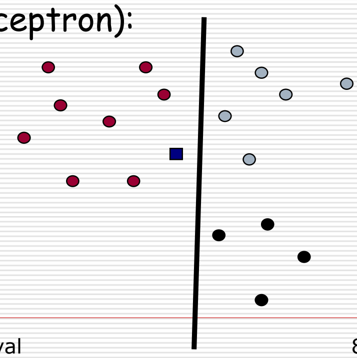
- A linear classifier is a classifier such that classification is performed by a dot product between the two vectors representing the document and the category, respectively. Therefore it consists in a document-like representation of the category c_i
- Linear classifiers are thus very efficient at classification time
- Methods for learning linear classifiers can be partitioned in two broad classes
  - Incremental methods (IMs) (or on-line) build a classifier soon after examining the first document, as incrementally refine it as they examine new ones
  - Batch methods (BMs) build a classifier by analyzing Tr all at once.

LCs for Binary Classification

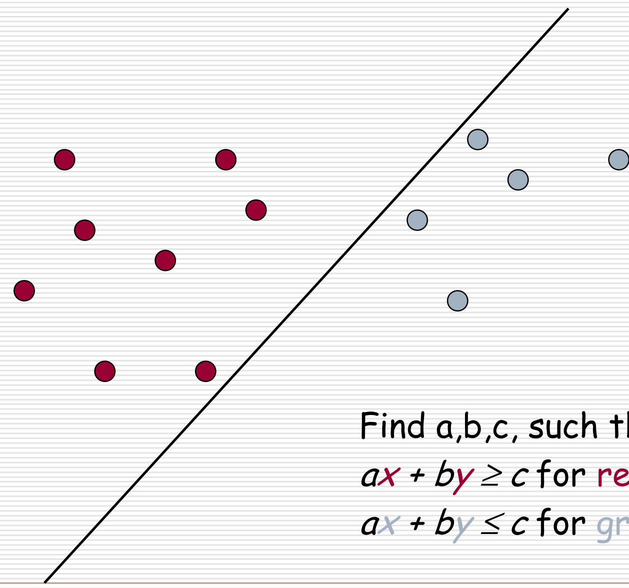
- Consider 2 class problems in VSM
  - Deciding between two classes, perhaps, sport and not sport
  - How do we define (and find) the separating surface (hyperplane)?
- How do we test which region a test doc is in?

Separation by Hyperplanes

- A strong high-bias assumption is *linear separability*:
  - in 2 dimensions, can separate classes by a line
  - in higher dimensions, need hyperplanes
- Can find separating hyperplane by *linear programming*
  (or can iteratively fit solution via perceptron):
  - separator can be expressed as
    $ax + by = c$



Linear programming / Perceptron



Find a,b,c, such that

$ax + by \geq c$ for red points

$ax + by \leq c$ for green points.

Linear programming / Perceptron

$w^t x + b > 0$	If x belongs to Sport (y = +1)
$w^t x + b < 0$	Otherwise (y = -1)

You can write $y([w\ b]\,[x\ 1]^t) > 0$

Thus, we want to compute a vector W s.t.

$$\boxed{y_i W^t X_i > 0}$$

where $W=[w\ b]^t$ and $X_i=[x_i\ 1]^t$

The Perceptron Algorithm

A simple (but powerful) method is the perceptron

1. Initialize W=0
2. For all training examples (X_i, y_i), i=1,..,n

   If (y_i $W^t \cdot X_i$ <= 0) $W = W + y_i X_i$
3. If no errors have been done in step 2, stop. Otherwise repeat step 2.

Linear classifier: Example

- Class: "interest" (as in interest rate)
- Example features of a linear classifier
-

w_i t_i	w_i t_i
• 0.70 prime	• -0.71 dlrs
• 0.67 rate	• -0.35 world
• 0.63 interest	• -0.33 sees
• 0.60 rates	• -0.25 year
• 0.46 discount	• -0.24 group
• 0.43 bundesbank	• -0.24 dlr

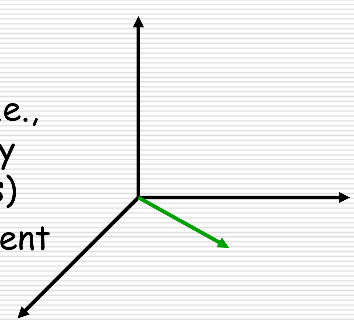
- To classify, find dot product of feature vector and weights
  - Score("rate discount dlrs world") = .67+.46-.71-.35=0.05 > 0
  - Score("prime dlrs") = .70-.71=-.01 < 0

Linear Classifiers

- Many common text classifiers are linear classifiers
  - Naïve Bayes
  - Perceptron
  - Rocchio
  - Logistic regression
  - Support vector machines (with linear kernel)
  - Linear regression
  - (Simple) perceptron neural networks
- 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?
  - Different training methods pick different hyperplanes
- Classifiers more powerful than linear often don't perform better. Why?

High Dimensional Data

- Documents are zero along almost all axes
- Most document pairs are very far apart (i.e., not strictly orthogonal, but only share very common words and a few scattered others)
- In classification terms: virtually all document sets are separable, for most any classification
- This is part of why linear classifiers are quite successful in this domain



Naive Bayes is a linear classifier

- Two-class Naive Bayes. We compute:

$$\log \frac{P(C \mid d)}{P(\overline{C} \mid d)} = \log \frac{P(C)}{P(\overline{C})} + \sum_{w \in d} \log \frac{P(w \mid C)}{P(w \mid \overline{C})}$$

- Decide class *C* if the odds ratio is greater than 1, i.e., if the log odds is greater than 0.
- So decision boundary is hyperplane:

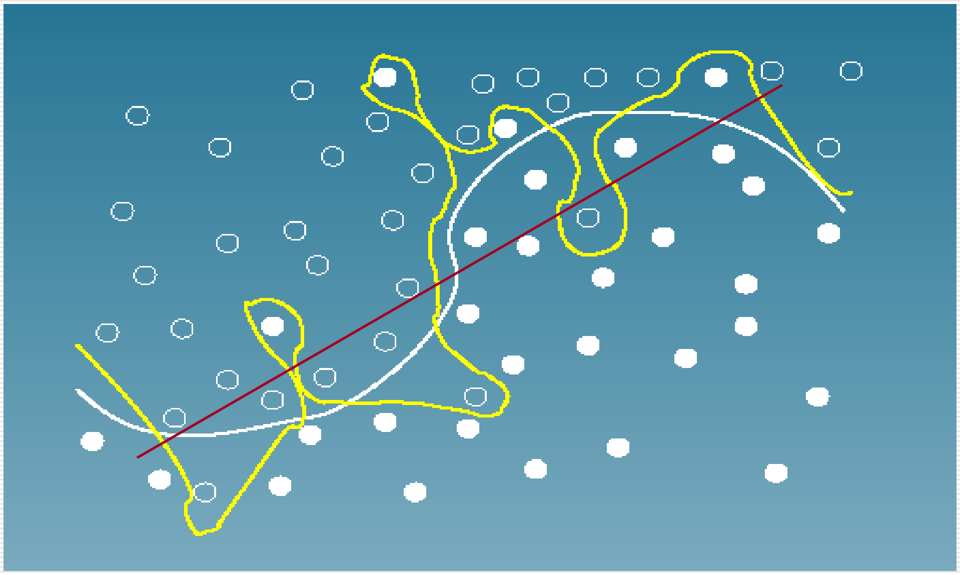
$$\alpha + \sum_{w \in V} \beta_w \times n_w = 0 \quad \text{where} \quad \alpha = \log \frac{P(C)}{P(\overline{C})};$$

$$\beta_w = \log \frac{P(w \mid C)}{P(w \mid \overline{C})}; \quad n_w = \#\text{ of occurrences of } w \text{ in } d$$

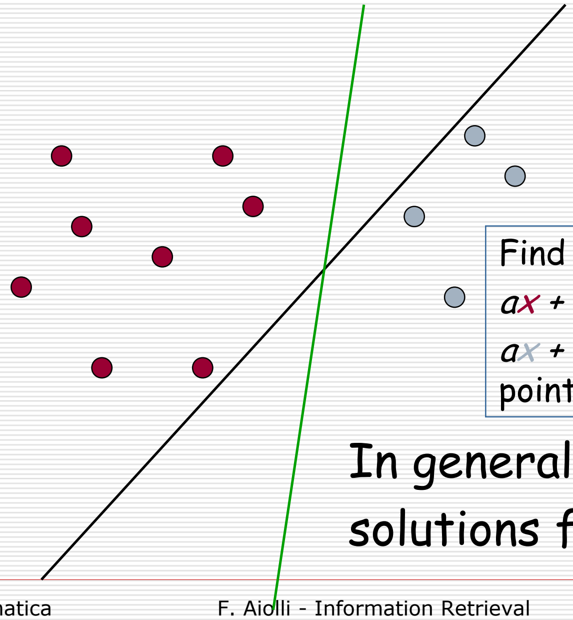
kNN vs. Linear Classifiers

- Bias/Variance tradeoff
  - Variance ≈ Capacity
- kNN has high variance and low bias.
  - Infinite memory
- LCs has low variance and high bias.
  - Decision surface has to be linear (hyperplane)
- Consider: Is an object a tree?
  - Too much capacity/variance, low bias
    - Botanist who memorizes
    - Will always say "no" to new object (e.g., # leaves)
  - Not enough capacity/variance, high bias
    - Lazy botanist
    - Says "yes" if the object is green
  - Want the middle ground

Bias vs. variance: Choosing the correct model capacity



Which Hyperplane?



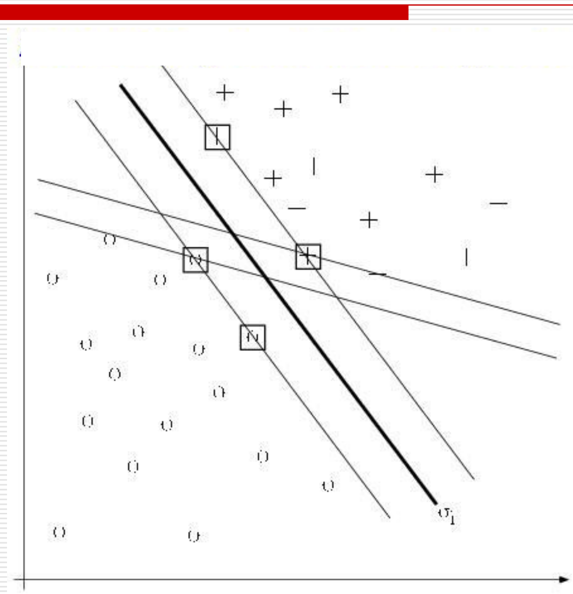
Find a,b,c, such that
$ax + by \geq c$ for red points
$ax + by \leq c$ for green points.

In general, lots of possible solutions for *a,b,c.*

The Support Vector Machine

- The support vector machine (SVM) method attempts to find, among all the decision surfaces $h_1, h_2, .., h_n$ in d-dimensional space, the one h_{svm} that does it by the widest possible margin
- This method applies the so called structural risk minimization principle, in contrast to the empirical minimization principle
- Learning a SVM is typically a quadratic problem

SVM



The Support Vector Machine

- The maximal margin hyperplane is also called optimal hyperplane
- Why it should be the best?
  - Keep training data far away from the classifier (fairly certain class. decisions)
  - The capacity of the model decreases as the separator become fatter. N.B. the bias has been fixed as we are looking for a linear separation in feature space

The Support Vector Machine

- The (functional) margin of data points is often used as a measure of confidence in the prediction of a classifier,

  $\rho_i = y_i (w\, x_i + b)$
- The geometric margin ρ of a classifier is the Euclidean distance between the hyperplane and the closest point
- It can be shown that $\rho = 2/||w||$

The SVM problem

- Find an hyperplane, consistent with the labels of the points, that maximizes the geometric margin, i.e.

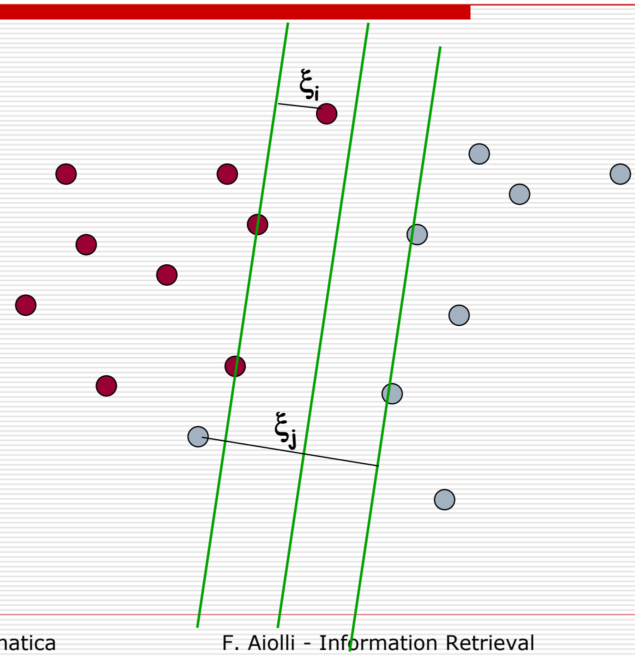
$$\text{Min } 1\backslash 2 \ ||w||^2$$
$$\text{And for all } \{(x_i, y_i),\ y_i(w\ x_i + b) \geq 1\}$$

- This is a (convex) constrained quadratic problem. Thus it guarantees a unique solution!
- Many QP algorithms exists to find the solution of this quadratic problem
- In SVM related literature, many algorithms have been devised ad hoc for this kind of quadratic problem (svmlight, bsvm, SMO, …)

Solving the optimization problem

- Typically, solving an SVM boils down into solving the dual problem where Lagrange multipliers $\alpha_i \geq 0$ are associated with every constraint in the primary problem
- The solution turns out to be in the form
  - $w = \sum_i y_i \alpha_i x_i$
  - $b = y_k - \langle w, x_k \rangle$ for any x_k s.t. $\alpha_k > 0$
- In the solution most of the α_i are zeros. Examples associated with non zero multipliers are called support vectors
- $h_{SVM}(x) = \text{sign}(w\ x + b) = \text{sign}(\sum_i y_i\ \alpha_i \langle x_i, x \rangle + b)$

Non-separable Datasets



Soft margin SVM

- Find an hyperplane, consistent with the labels of the points, that maximizes the function

$$\text{Min } 1\backslash 2\ ||w||^2 + C \sum_i \xi_i$$

$$\text{And for all } \{(x_i,y_i),\ y_i(w\ x_i + b) \geq 1-\xi_i,\ \xi_i \geq 0\}$$

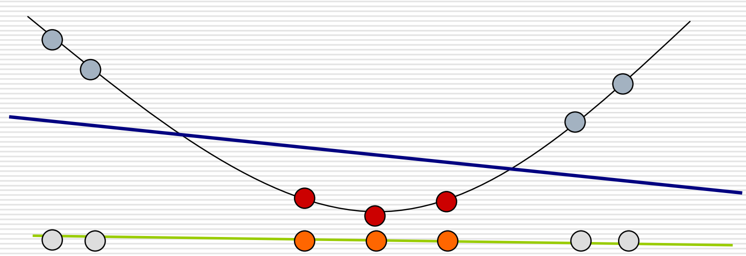
- The parameter C can be seen as a way to control overfitting.
- As C becomes larger it is unactractive to not respect the data at the cost of reducing the geometric margin.
- When C is small, larger margin is possible at the cost of increasing errors in training data
- Interestingly, the SVM solution is in the same form as in the hard margin case!

Non-separable Datasets

How can we separate these data?

Non-separable Datasets

Projecting them into a higher dimensional space



$$\Phi : \mathbf{x} \rightarrow \phi(\mathbf{x})$$

Solving the optimization problem

- $h_{SVM}(x) = \text{sign}(w\ \phi(x) + b)$
  $= \text{sign}(\sum_i y_i\ \alpha_i\ \langle\phi(x_i), \phi(x)\rangle + b)$
  $= \text{sign}(\sum_i y_i\ \alpha_i\ K(x_i,x) + b)$

- Where $K(x_i,x)$ is the kernel function such that $K(x_i,x) = \langle\phi(x_i), \phi(x)\rangle$

Advantages of SVM

- SVMs have important advantages for TC
  - The 'best' decision surface is determined by only a small set of training examples, called the support vector (in the linear case, everything can be compacted in one vector only)
  - Different kernel functions can be plugged in, corresponding to different ways of computing the similarity of document
  - The method is applicable also to the case in which the sample is not separable
  - No term selection is usually needed, as SVMs are fairly robust to overfitting and can scale up to high dimensionalities
- SVM has been shown among the top performing systems in a number of experiments [Dumais+98,Joachims98,Yang&Liu99]

The (DUAL) Perceptron Algorithm

1. Initialize α_i=0, i=1,…,n

2. For all training examples (X_i,y_i), i=1,..,n
   If (y_i $\sum_{j=1,..,n} y_j \alpha_j X_i X_j$ <= 0) α_i++

3. If no errors have been done in step 2, stop. Otherwise repeat step 2.

Committees of Classifiers

- Classifier Committee (CCs) is based on applying k different classifiers h_1, .. , h_k to the same task and then combining their outcomes
- Usually the classifiers are chosen to be different in some respect
  - Different indexing approach
  - Different learning method applied
  - Different types of errors !!
- It must be defined a way to combine them
- Justified only by superior effectiveness

Combination rules

- **Majority Voting**: the classification decision that reach the majority of votes is taken
- **Weighted Linear Combination**: a weighted sum of the k CSV_i's yields the final CSV_i
- **Dynamic Classifier Selection**: the judgment of the classifier h_t that yields the best effectiveness on the validation examples most similar to d_j is adopted
- **Adaptive Classifier Combination**: the judgment of all the classifiers are summed together, but their individual contribution is weighted by their effectiveness on the examples most similar to d_j

Boosting

- Boosting is a CC method whereby the classifiers ('weak hypothesis') are trained sequentially by the same learner ('weak learner'), and are combined into a CC ('final hypothesys')
- The training of h_t is done in such a way to try to make the classifier to perform well on examples in which $h_1,..,h_{t-1}$ have performed worst
- AdaBoost is a popular Boosting algorithm

Freund & Schapire's AdaBoost

At iteration s:

1. Passes a distribution D_s of weights to the weak learner, where $D_s(d_j)$ measures how effective $h_1,..,h_{s-1}$ have been in classifying d_j
2. The weak learner returns a new weak hypothesis h_s that concentrates on documents with the highest D_s values
3. Runs h_s on Tr and uses the results to produce an updated distribution D_{s+1} where
   - Correctly classified documents have their weights decreased
   - Misclassified documents have their weights increased

Evaluating TC systems

- Similarly to IR systems, the evaluation of TC systems is to be conducted experimentally, rather than analytically
- Several criteria of quality:
  - Training-Time efficiency
  - Classification-Time efficiency
  - Effectiveness
- In operational situations, all three criteria must be considered, and the right tradeoff between them depends on the application