Text Mining, Information and Fact Extraction
Part 2: Machine Learning Techniques

Marie-Francine Moens
Department of Computer Science
Katholieke Universiteit Leuven, Belgium
sien.moens@cs.kuleuven.be

Problem

- Symbolic techniques are useful in very restricted subject domains that exhibit standard patterns:
  - e.g., mining of a weather report
- In most settings language is characterized by:
  - a variety of patterns that express the same or similar meanings
  - ambiguous patterns that receive their meaning based on the context
- Patterns change in time (e.g., blog and chat languages)

Problem

- ⇒ manual effort is huge to build all the needed (contextual) patterns for all kinds of information extraction tasks
- IE in terrorism domain: experiment of Riloff (1996): automatic construction of dictionary of extraction patterns from an annotated training corpus achieved 98% of the performance of handcrafted patterns

=> **machine learning of extraction patterns**

[Riloff AI 1996]

Problem

- Since 1996: interest in machine learning for information extraction:
  - Usually supervised learning algorithms:
    - e.g., learning of rules and trees, support vector machines, maximum entropy classification, hidden Markov models, conditional random fields
  - Unsupervised learning algorithms:
    - e.g., clustering (e.g., noun phrase coreferent resolution)
  - Weakly supervised learning algorithms

Overview

- Features
- Supervised methods:
  - Support vector machines and kernel methods
  - Probabilistic models:
    - Naive Bayes models
    - Maximum entropy models

(exemplary results are added)

Symbols used

- $\boldsymbol{x}$ = object (to be) classified
- y = class label (to be) assigned to $\boldsymbol{x}$

Generative versus discriminative classification

- In classification: given inputs $\boldsymbol{x}$ and their labels y:
  - **Generative classifier** learns a model of the joint probability $p(\boldsymbol{x},y)$ and makes its predictions by using Bayes' rule to calculate $p(y|\boldsymbol{x})$ and then selects the most likely label y:
    - e.g., Naive Bayes, hidden Markov model
    - to make computations tractable: simplifying independence assumptions

Generative versus discriminative classification

- **Discriminative classifier** is trained to model the conditional probability $p(y|\boldsymbol{x})$ directly and selects the most likely label y, or learns a direct map from inputs $\boldsymbol{x}$ to the class labels:
  - e.g., maximum entropy model, support vector machine
  - better suited when including rich, overlapping features

Maximum entropy principle

- Text classifiers are often trained with **incomplete information**
- Probabilistic classification can adhere to the principle of maximum entropy: When we make inferences based on incomplete information, we should draw them from that probability distribution that has the maximum entropy permitted by the information we have: e.g.,
  - maximum entropy model, conditional random fields

Combining classifiers

- Multiple classifiers are learned and combined:
  - **Bagging**: e.g., sample of size *n* is taken randomly with replacement from the original set of *n* training documents
    - Adaptive resampling: no random sampling, but objective = to increase the odds of sampling documents that previously induced classifiers have erroneously been classified
  - **Stacking**: predictions from different classifiers are used as input for a meta-learner
  - **Boosting**: generating a sequence of classifiers, after each classification greater weights are assigned to objects with an uncertain classification, and classifier is retrained

Feature selection and extraction

- In classification tasks: object is described with set of **attributes** or **features**
- Typical features in text classification tasks:
  - word, phrase, syntactic class of a word, text position, the length of a sentence, the relationship between two sentences, an *n*-gram, a document (term classification), ….
  - choice of the features is application- and domain-specific
- Features can have a value, for text the value is often:
  - numeric, e.g., discrete or real values
  - nominal, e.g. certain strings
  - ordinal, e.g., the values 0= small number, 1 = medium number, 2 = large number

Feature selection and extraction

- The features together span a multi-variate space called the measurement space or feature space:
  - an object $\boldsymbol{x}$ can be represented as:
    - a **vector of features**:

      $\boldsymbol{x} = [x_1, x_2, \ldots, x_p]^T$

      where p = the number of features measured
    - as a **structure**: e.g.,
      - representation in first order predicate logic
      - graph representation (e.g., tree) where relations between features are figured as edges between nodes and nodes can contain attributes of features

Feature selection

- = eliminating low quality features:
  - redundant features
  - noisy features
- decreases computational complexity
- decreases the danger of overfitting in supervised learning (especially when large number of features and few training examples)
- increases the chances of detecting valuable patterns in unsupervised learning and weakly supervised learning
- **Overfitting:**
  - the classifier perfectly fits the training data, but fails to generalize sufficiently from the training data to correctly classify the new case

Feature selection

- In supervised learning:
  - feature selection often incorporated in training algorithms:
    - incrementally add features, discard features, or both, evaluating the subset of features that would be produced by each change (e.g., algorithms that induce decision trees or rules from the sample data)
  - feature selection can be done after classification of new objects:
    - by measuring the error rate of the classification
    - those features are removed from or added to the feature set when this results in a lower error rate on the test set

Feature extraction

- = creates new features by applying a set of operators upon the current features:
  - a single feature can be replaced by a new feature (e.g., replacing words by their stem)
  - a set of features is replaced by one feature or another set of features
    - use of logical operators (e.g., disjunction), arithmetical operators (e.g., mean, dimensionality reduction)
  - choice of operators: application- and domain-specific
- In supervised learning can be part of training or done after classification of new objects

Support vector machine (SVM)

- Discriminant analysis:
  - determining a function that in a best way discriminates between two classes
  - text categorization: two classes: positive and negative examples of a class
  - e.g., linear discriminant analysis finds a linear combination of the features (variables) : hyperplane (line in two dimensions, plane in three dimensions, etc.) in the *p*-dimensional feature space that best separates the classes
  - usually, there exist different hyperplanes that separate the examples of the training set in positive and negative ones

Support vector machine

- **Support vector machine**:
  - when two classes are linearly separable:
    - find a hyperplane in the *p*-dimensional feature space that best separates with **maximum margins** the positive and negative examples
    - maximum margins: with maximum Euclidean distance (= margin *d*) to the closest training examples (**support vectors**)
    - e.g., decision surface in two dimensions
  - idea can be generalized to examples that are not necessarily linearly separable and to examples that cannot be represented by linear decision surfaces

Support vector machine

- **Linear support vector machine**:
  - case: **trained on data that are separable** (simple case)
  - input is a set of *n* training examples:

$$S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$$

where $\boldsymbol{x}_i \in \Re^p$ and $y_i \in \{-1,+1\}$ indicating that $\boldsymbol{x}_i$ is a negative or positive example respectively

Support vector machine

Suppose we have some hyperplane which separates the positive from the negative examples, the points which lie on the hyperplane satisfy:

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b = 0$$

where $\boldsymbol{w}$ = normal to the hyperplane

$\frac{|b|}{\|\boldsymbol{w}\|}$ = perpendicular distance from the hyperplane to the origin

$\|\boldsymbol{w}\|$ = Euclidean norm of $\boldsymbol{w}$

let d_+ (d_-) be the shortest distance from the separating hyperplane to the closest positive (negative) example

define the margin of the separating hyperplane to be d_+ and d_-

search the hyperplane with largest margin

Given separable training data that satisfy the following constraints:

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b \geq +1 \qquad \text{for } y_i = +1 \qquad (1)$$

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b \leq -1 \qquad \text{for } y_i = -1 \qquad (2)$$

which can be combined in 1 set of inequalities:

$$y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1 \geq 0 \qquad \text{for } i = 1, \ldots, n \qquad (3)$$

The hyperplane that defines one margin is defined by:

$$H_1 : \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b = 1$$

with normal $\boldsymbol{w}$ and perpendicular distance from the origin

$$\frac{|1 - b|}{\|\boldsymbol{w}\|}$$

The hyperplane that defines the other margin is defined by:

$$H_2 : \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b = -1$$

with normal $\boldsymbol{w}$ and perpendicular distance from the origin

$$\frac{|-1 - b|}{\|\boldsymbol{w}\|}$$

Hence $d_+ = d_- = \frac{1}{\|\boldsymbol{w}\|}$ and the margin = $\frac{2}{\|\boldsymbol{w}\|}$

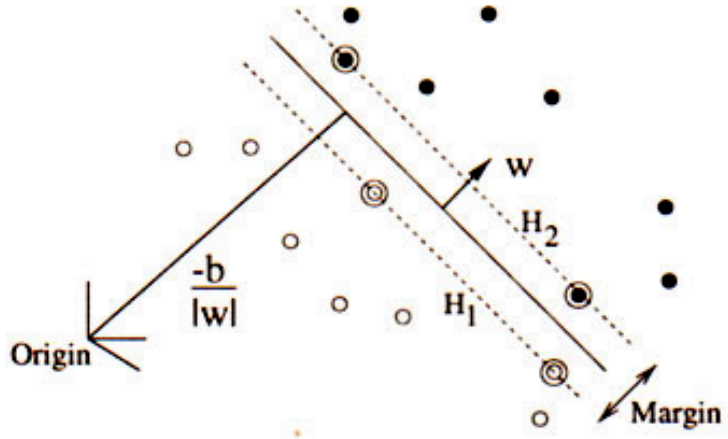

Figure 5. Linear separating hyperplanes for the separable case. The support vectors are circled.

[Burges 1995]

Hence we assume the following objective function to maximize the margin:

$$\text{Minimize}_{\boldsymbol{w},b} \left\langle \boldsymbol{w} \cdot \boldsymbol{w} \right\rangle$$

$$\text{Subject to } y_i(\left\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \right\rangle + b) - 1 \geq 0, \quad i = 1,...,n$$

A dual representation is obtained by introducing Lagrange multipliers λ_i, which turns out to be easier to solve:

$$\text{Maximize } W(\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{n} \lambda_i \lambda_j y_i y_j \langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle \qquad (4)$$

$$\text{Subject to: } \lambda_i \geq 0$$

$$\sum_{i=1}^{n} \lambda_i y_i = 0 , \quad i = 1,...,n$$

Yielding the following decision function:

$$h(\boldsymbol{x}) = sign(f(\boldsymbol{x}))$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \lambda_i y_i \langle \boldsymbol{x}_i \cdot \boldsymbol{x} \rangle + b \qquad (5)$$

The decision function only depends on support vectors, i.e., for which $\lambda_i > 0$. Training examples that are not support vectors have no influence on the decision function

Support vector machine

- **Trained on data not necessarily linearly separable (soft margin SVM)**:
  - the amount of training error is measured using slack variables ξ_i the sum of which must not exceed some upper bound
  - The hyperplanes that define the margins are now defined as:

$$H_1 : \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b = 1 - \xi_i$$

$$H_2 : \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b = -1 + \xi_i$$

- Hence we assume the following objective function to maximize the margin:

$$\text{Minimize}_{\xi, \boldsymbol{w}, b} \left\langle \boldsymbol{w} \cdot \boldsymbol{w} \right\rangle + C \sum_{i=1}^{n} \xi_i^2$$

$$\text{Subject to } y_i(\left\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \right\rangle + b) - 1 + \xi_i \geq 0 \ , \quad i = 1,\ldots,n$$

where $\sum_{i=1}^{n} \xi_i^2$ = penalty for misclassification

C = weighting factor

The decision function is computed as in the case of data objects that are linearly separable (cf. 5)

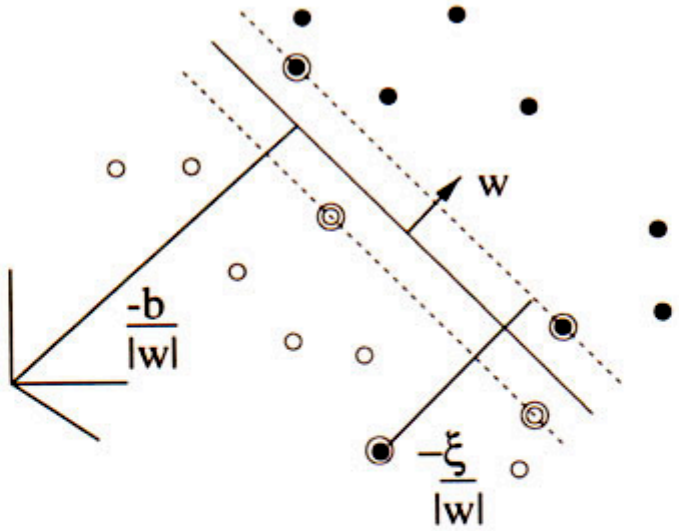

Figure 6. Linear separating hyperplanes for the non-separable case.

[Burges 1995]

Support vector machine

- When classifying natural language data, it is not always possible to linearly separate the data: in this case we can map them into a feature space where they are linearly separable
- Working in a high dimensional feature space gives computational problems, as one has to work with very large vectors
- In the dual representation the data appear only inside inner products (both in the training algorithm shown by (4) and in the decision function of (5)): in both cases a kernel function can be used in the computations

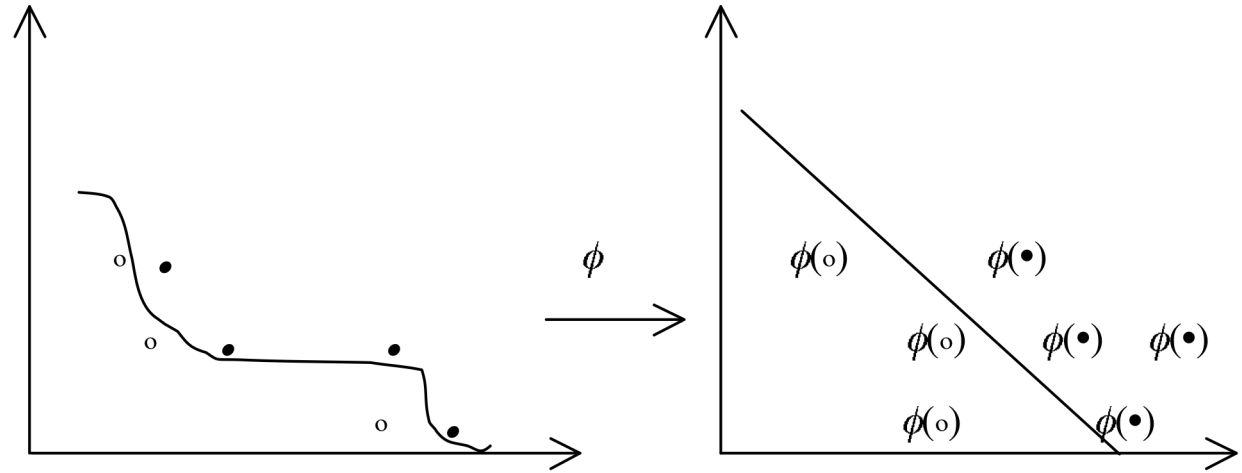

Fig. 5.2. A mapping of the features can make the classification task more easy (after Christianini and Shawe-Taylor 2000).

Kernel function

- A kernel function K is a mapping K: S x $S \rightarrow [0, \infty]$ from the instance space of training examples S to a similarity score:

$$K(\boldsymbol{x}_i,\boldsymbol{x}_j) = \sum_k \phi_k(\boldsymbol{x}_i)\phi_k(\boldsymbol{x}_j) = \left\langle \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j) \right\rangle$$

- In other words a kernel function is an inner product in some feature space
- The kernel function must be:
  - symmetric $[K(\boldsymbol{x}_i,\boldsymbol{x}_j) = K(\boldsymbol{x}_j,\boldsymbol{x}_i)]$
  - positive semi-definite: if $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n \in S$, then the n x n matrix G (*Gram matrix or kernel matrix*) defined by $G_{ij} = K(\boldsymbol{x}_i,\boldsymbol{x}_j)$ is positive semi-definite*

* has non-negative eigenvalues

Support vector machine

- The decision function $f(\mathbf{x})$ we can just replace the dot products with kernels $K(\boldsymbol{x}_i,\boldsymbol{x}_j)$:

$$h(\boldsymbol{x}) = sign(f(\boldsymbol{x}))$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \lambda_i y_i \langle \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}) \rangle + b$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \lambda_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

Support vector machine

- Advantages:
  - SVM can cope with many (noisy) features: no need for a priori feature selection, though you might select features for reasons of efficiency
  - many text categorization problems are linearly separable

Kernel functions

- Typical kernel functions: linear (mostly used in text categorization), polynomial, radial basis function (RBF)
- In natural language: data are often structured by the modeling of relations=> kernels that (efficiently) compare structured data: e.g.,
  - Rational kernels = similarity measures over sets of sequences
    - n-gram kernels
    - convolution kernels
  - Kernels on trees

n-gram kernels

- *n*-gram is a block of adjacent characters from an alphabet Σ
- *n*-gram kernel: to compare sequences by means of subsequences they contain:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{s \in \Sigma^n} \#(s \in \boldsymbol{x}_i)\ \#(s \in \boldsymbol{x}_j)$$

where $\#(s \in \boldsymbol{x})$ denotes the number of occurrences of s in $\boldsymbol{x}$

- the kernel function can be computed in $O(|\boldsymbol{x}_i| + |\boldsymbol{x}_j|)$ time and memory by means of a special suited data structure allowing one to find a compact representation of all subsequences in $\boldsymbol{x}$ in only $O(|\boldsymbol{x}|)$ time and space **[Vishwanathan & Smola 2004]**

Convolution kernels

- Object $x \in \mathcal{X}$ consists of substructures $x_p \in \mathcal{X}_p$ where $1 \leq p \leq r$ and r denotes the number of overall substructures
- Given the set of all possible substructures $\mathcal{P}(\mathcal{X})$, one can define a relation R (e.g., part of) between a subset of $\mathcal{P}$ and the composite object x
- Given a finite number of subsets, R is called finite
- Given a finite relation R, R^{-1} defines the set of all possible decompositions of x into its substructures: $R^{-1}(x) = \{z \in \mathcal{P}(\mathcal{X}) : R(z, x)\}$

Convolution kernels

- The *R*-convolution kernel:

$$K(\boldsymbol{x},\boldsymbol{y}) = \sum_{\boldsymbol{x}'\in R^{-1}(\boldsymbol{x})} \sum_{\boldsymbol{y}'\in R^{-1}(\boldsymbol{y})} \prod_{i=1}^{r} K_i(\boldsymbol{x}'_i, \boldsymbol{y}'_i)$$

  is a valid kernel with K_i being a positive semi-definite kernel on $\mathcal{X}_i$

- The idea of decomposing a structured object into parts can be applied recursively so that one only requires to construct kernels over the "atomic" parts of $\mathcal{X}_i$

Tree kernels

- Based on convolution kernels
- A tree is mapped into its sets of subtrees, the kernel between two trees $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is then computed by taking the weighted sum of all terms between both trees
- Efficient algorithms to compute the subtrees:
  - $O(|\boldsymbol{x}_i|.|\boldsymbol{x}_j|)$, where $|\boldsymbol{x}|$ is the number of nodes of the tree
  - $O(|\boldsymbol{x}_i|+|\boldsymbol{x}_j|)$, when restricting the sum to all proper rooted subtrees

[Collins & Duffy 2004] [Vishwanathan & Smola 2004]

Tree kernels

- Comparison of augmented dependency trees for relation extraction from sentences:
  - recursive function to combine parse tree similarity with similarity $s(\boldsymbol{x}_i, \boldsymbol{x}_j)$ based on feature correspondence of the nodes of the trees

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \begin{cases} 0, & \text{if } m(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0 \\ s(\boldsymbol{x}_i, \boldsymbol{x}_j) + K_c(\boldsymbol{x}_i[c], \boldsymbol{x}_j[c]) & \text{otherwise} \end{cases}$$

where $m(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \{0, 1\}$ determines whether two nodes are matchable or not

c = children subtrees

Feature	Example
word	*troops, Tikrit*
part-of-speech (*24 values*)	NN, NNP
general-pos (*5 values*)	noun, verb, adj
chunk-tag	NP, VP, ADJP
entity-type	person, geo-political-entity
entity-level	name, nominal, pronoun
Wordnet hypernyms	*social group, city*
relation-argument	ARG_A, ARG_B

Table 3: List of features assigned to each node in the dependency tree.

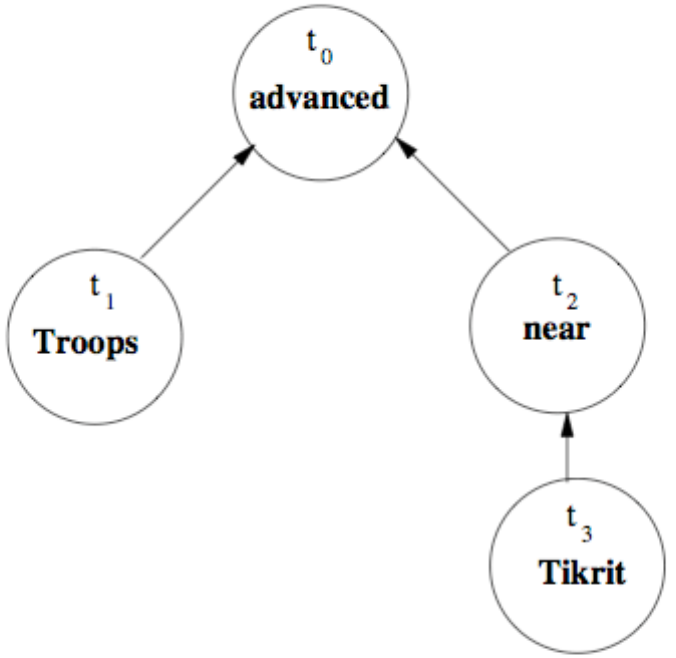

Figure 1: A dependency tree for the sentence *Toops advanced near Tikrit*.

[Culotta & Sorensen 2004]

Naive Bayes (NB) model

- Bayesian classifier:
  - the posterior probability that a new, previously unseen object belongs to a certain class given the features of the object is computed:
    - based on the probabilities that these individual features are related to the class

- **Naive** Bayes classifier:
  - computations simplified by the assumption that the features are conditionally independent

Naive Bayes model

$$P(C_j|w_1,\ldots,w_p) = \frac{P(w_1,\ldots,w_p|C_j)P(C_j)}{P(w_1,\ldots,w_p)}$$

where $w_1,\ldots, w_p$ = set of *p* features

ranking

$$P(C_j|w_1,\ldots,w_p) = P(w_1,\ldots,w_p|C_j)P(C_j)$$

independence assumption

$$= P(C_j)\prod_{i=1}^{p} P(w_i|C_j)$$

practical implementation

$$= \log P(C_j) + \sum_{i=1}^{p} \log P(w_i|C_j)$$

Naive Bayes model

normalized form

$$P(C_j|w_1,\ldots,w_p) = P(C_j)\frac{P(C_j)\prod_{i=1}^{p} P(w_i|C_j)}{\sum_{k=1}^{|C|} P(C_k)\prod_{i=1}^{p} P(w_i|C_k)}$$

Naive Bayes model

- Estimations from training set:
  - $P(C_j)$
  - $P(w_i|C_j)$:
    - **binomial or Bernoulli model**:
      - fraction of objects of class C_j in which feature w_i occurs
    - **multinomial model**:
      - fraction of times that feature w_i occurs across all objects of class C_j
      - additional positional independence assumptions
  - To avoid zero probabilities: add one to each count (add-one or Laplace smoothing)

TRAINMULTINOMIALNB($\mathbb{C}$, $\mathbb{D}$)

1 $V \leftarrow$ EXTRACTVOCABULARY($\mathbb{D}$)
2 $N \leftarrow$ COUNTDOCS($\mathbb{D}$)
3 **for each** $c \in \mathbb{C}$
4 **do** $N_c \leftarrow$ COUNTDOCSINCLASS($\mathbb{D}$, c)
5 $prior[c] \leftarrow N_c/N$
6 $text_c \leftarrow$ CONCATENATETEXTOFALLDOCSINCLASS($\mathbb{D}$, c)
7 **for each** $t \in V$
8 **do** $T_{ct} \leftarrow$ COUNTTOKENSOFTERM($text_c$, t)
9 **for each** $t \in V$
10 **do** $condprob[t][c] \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$
11 **return** V, $prior$, $condprob$

APPLYMULTINOMIALNB($\mathbb{C}$, V, $prior$, $condprob$, d)

1 $W \leftarrow$ EXTRACTTOKENSFROMDOC(V, d)
2 **for each** $c \in \mathbb{C}$
3 **do** $score[c] \leftarrow \log prior[c]$
4 **for each** $t \in W$
5 **do** $score[c] \mathrel{+}= \log condprob[t][c]$
6 **return** $\arg\max_{c \in \mathbb{C}} score[c]$

Figure 13.2 Naive Bayes algorithm (multinomial model): Training and testing.

[Manning et al. 2008]

Table 13.3 Multinomial versus Bernoulli model.

	multinomial model	Bernoulli model
event model	generation of token	generation of document
random variable(s)	$X = t$ iff t occurs at given pos	$U_t = 1$ iff t occurs in doc
document representation	$d = \langle t_1, \ldots, t_k, \ldots, t_{n_d} \rangle$, $t_k \in V$	$d = \langle e_1, \ldots, e_i, \ldots, e_M \rangle$, $e_i \in \{0, 1\}$
parameter estimation	$\hat{P}(X = t\|c)$	$\hat{P}(U_i = e\|c)$
decision rule: maximize	$\hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(X = t_k\|c)$	$\hat{P}(c) \prod_{t_i \in V} \hat{P}(U_i = e_i\|c)$
multiple occurrences	taken into account	ignored
length of docs	can handle longer docs	works best for short docs
# features	can handle more	works best with fewer
estimate for term the	$\hat{P}(X = \text{the}\|c) \approx 0.05$	$\hat{P}(U_{\text{the}} = 1\|c) \approx 1.0$

[Manning et al. 2008]

Naive Bayes model

- Class assignment:
  - find the *k* most probable classes; *k* = 1: $\arg\max_{C_j} P\left(C_j \middle| w_1,\ldots,w_p\right)$
  - alternative: select classes for which $P\left(C_j \middle| w_1,\ldots,w_p\right) > \text{threshold}$
- Advantages:
  - Efficiency
- Disadvantages:
  - Independence assumptions
  - No accurate probability estimates: close to 0; winning class after normalization close to 1

Maximum entropy principle

- Used in classifiers that compute a probabilistic class assignment
- **Maximum entropy principle**: given a set of training data, model what is known and assume no further knowledge about the unknowns by assigning them equal probability
- In other words we choose the model p^* that preserves as much uncertainty as possible between all the models $p \in P$ that satisfy the constraints enforced by the training examples
- Examples:
  - maximum entropy model
  - conditional random field

p(a,b)	0	1	
x	?	?	
y	?	?	
Total	.6		1

Table 1: The task is to find a probability distribution p under constraints $p(x,0) + p(y,0) = .6$.

	0	1	
x	.5	.1	
y	.1	.3	
Total	.6		1

	0	1	
x	.3	.2	
y	.3	.2	
Total	.6		1

Table 1: a) One way to satisfy the constraints; b)The most "uncertain" way to satisfy the constraints.

Maximum entropy model

- Given *n* training examples $S = \{(\boldsymbol{x}, y)_1, \ldots, (\boldsymbol{x}, y)_n\}$.
  where $\boldsymbol{x}$ = feature vector and y = class
- We choose the model p^* that preserves as much uncertainty as possible, or which maximizes the entropy $H(p)$ between all the models $p \in P$ that satisfy the constraints enforced by the training examples:

$$H(p) = -\sum_{(x,y)} p(x,y) \log p(x,y)$$

$$p^* = \arg\max_{p \in P} H(p)$$

Maximum entropy model

- The training data are described with *k* feature functions*:

$$f_j(\boldsymbol{x},y)=\begin{cases}1 & \text{if } (\boldsymbol{x},y) \text{ satisfies a certain constraint}\\ 0 & \text{otherwise}\end{cases}$$

e.g.,
$$f_j(\boldsymbol{x},y)=\begin{cases}1 & \text{if } x_1 = \textit{Lou Gerigh} \text{ and } y = \textit{disease}\\ 0 & \text{otherwise}\end{cases}$$

$$f_j(\boldsymbol{x},y)=\begin{cases}1 & \text{if } x_2 = \textit{say} \text{ and } y = \textit{person}\\ 0 & \text{otherwise}\end{cases}$$

* Are usually binary-valued because of efficient training

Maximum entropy model

- The statistics of a feature function is captured by ensuring that the model adheres to the following equality:

$$E_p(f_j) = E_{\tilde{p}}(f_j)$$

where

$$E_p(f_j) = \sum_{\boldsymbol{x},y} \tilde{p}(\boldsymbol{x})\, p(y|\boldsymbol{x})\, f_j(\boldsymbol{x},y) \qquad (1)$$

(approximated)

= expectation of the feature function f_j

$$E_{\tilde{p}}(f_j) = \sum_{\boldsymbol{x},y} \tilde{p}(\boldsymbol{x},y)\, f_j(\boldsymbol{x},y) \qquad (2)$$

= empirical expectation of the feature function f_j

Maximum entropy model

- The principle of maximum entropy recommends that we use p^*:

$$P = \{p \mid E_p(f_j) = E_{\tilde{p}}(f_j), \quad j = 1,\ldots,k\}$$

$$p^* = \arg\max_{p \in P} H(p)$$

It has been shown that $p^*(S)$ is unique and must be in the following form:

$$p^*(y|\boldsymbol{x}) = \frac{1}{Z}\exp\left(\sum_{j=1}^{k} \lambda_j f_j(\boldsymbol{x}, y)\right), \quad 0 < \lambda_j < \infty \qquad (3)$$

where

$f_j(\boldsymbol{x}, y)$ = one of the k binary-valued feature functions

λ_j = parameter adjusted to model the observed statistics

Z = normalizing constant

Maximum entropy model: training

The empirical distribution (2) is estimated from the training data via maximum likelihood estimation:

$E_{\tilde{p}}(f_j) = \tilde{p}$ (x_2 = "say", y = person)

≈ count (x_2 = "say" and y = person)/*N* where *N* is the total number of training events

The model must adjust the parameter λ_j such that its expectation of f_j matches the empirical one, while simultaneously matching the rest of the feature functions with their expectations

The model estimates $p^*(S)$ by adjusting the *k* model parameters λ_j, $1 <= j <= k$, subject to the constraints of the *k* feature functions

e.g., by the generalized iterative scaling algorithm

Maximum entropy model

Generalized iterative scaling:

input: feature functions: $f_1, \ldots, f_k$, empirical distribution $E_{\tilde{p}}(f_j)$

output: for the feature functions: the optimal parameter values λ_j

The algorithm requires that the sum of the features for each possible (x,y) is equal to a constant C, which is set at the greatest possible feature sum and therefore adds a feature as:

$$f_{k+1}(\boldsymbol{x}, y) = C - \sum_{j=1}^{k} f_j(\boldsymbol{x}, y)$$

Initialize $\{\lambda_j^{(1)}\}$, usually we choose $\lambda_j^{(1)} = 1$, $1 <= j <= k+1$

Maximum entropy model

Iteration:

1. Compute $p^{(l)}(x,y)$ for the distribution $p^{(l)}$ given by the $\{\lambda_j^{(l)}\}$, for each element (x,y) in the training set with (3) taking into account k+1 features

2. Compute according to (1) $E_{p^{(l)}}(f_j)$ $1<=j<=k+1$

3. Update the parameters λ_j : $$\lambda_j^{(l+1)} = \lambda_j^{(l)} \left(\frac{E_{\tilde{p}} f_j}{E_{p^{(l)}} f_j} \right)^{\frac{1}{C}}$$

Iteration stops: when the differences between $\lambda_j^{(l)}$ and $\lambda_j^{(l+1)}$ are very small

Maximum entropy model

- To classify new information unit with feature vector $\boldsymbol{x}$: compute (3) for each possible class y and choose y with greatest probability

- Advantages:
  - good results:
    - when dependencies exist between the features
    - with incomplete training data
  - no need for a priori feature selection

Maximum entropy model

- For example, used in named entity recognition:
  - = assignment of a semantic class to a named entity in the text
  - often restricted to the classification of proper names: i.e., as persons, companies, locations, currencies, ...
  - maximum entropy classification gives good results even with a limited number of training examples, e.g., MENERGI-system in next example

Systems	Size of training data	F-measure
SRA '95	Hand-coded	96.4%
IdentiFinder '99	650,000 words	94.9%
MENERGI	160,000 tokens	93.27%
IdentiFinder '99 (from graph)	> 200,000 words	About 93%
IdentiFinder '97	450,000 words	93%
IdentiFinder '97	about 100,000 words	91%-92%

Table 5: Comparison of results for MUC-6

Systems	Size of training data	F-measure
LTG system '98	Hybrid hand-coded	93.39%
IdentiFinder '98	790,000 words	90.44%
MENE + Proteus '98	Hybrid hand-coded 321,000 tokens	88.80%
MENERGI	180,000 tokens	87.24%
MENE+reference-resolution '99	321,000 tokens	86.56%
MENE '98	321,000 tokens	84.22%

[Chieu & Ng COLING. 2002]

Table 6: Comparison of results for MUC-7

- For example, used in temporal relation recognition:
  - Problem of **data sparseness**: here use of temporal reasoning to artificially expand the amount of training data ("CLOSED" in next slide): e.g.,

    If relation (A,B) = BEFORE && relation (B,C) = INCLUDES

    then infer relation (A,C) = BEFORE

	UNCLOSED (ME)						CLOSED (ME-C)					
	Event-Event			Event-Time			Event-Event			Event-Time		
Accuracy:	62.5 (51.6)			76.13 (65.3)			93.1 (75.2)			88.25 (62.3)		
Relation	Prec	Rec	F	Prec	Rec	F	Prec	Rec	F	Prec	Rec	F
IBEFORE	50.00	27.27	35.39	0	0	0	77.78	60.86	68.29	0	0	0
BEGINS	50.00	41.18	45.16	60.00	50.00	54.54	85.25	82.54	83.87	76.47	74.28	75.36
ENDS	94.74	66.67	78.26	41.67	27.78	33.33	87.83	94.20	90.90	79.31	77.97	78.62
SIMULTANEOUS	50.35	50.00	50.17	33.33	20.00	25.00	62.50	38.60	47.72	73.68	56.00	63.63
INCLUDES	47.88	34.34	40.00	80.92	62.72	84.29	90.41	88.23	89.30	86.07	80.78	83.34
BEFORE	68.85	79.24	73.68	70.47	62.72	66.37	94.95	97.26	96.09	90.16	93.56	91.83

Table 2. Machine learning results using unclosed and closed data

[Mani et al. COLING-ACL 2006]

References

Bakir G.H., Hofmann, T.,Schölkopf, B., Smola, A.J., Taskar, B. & Vishwanathan, S.V.N. (2007) (Eds.), *Predicting Structured Data.* Cambridge, MA: MIT Press.

Berger, Adam, Stephen A. Della Pietra and Vincent J. Della Pietra (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22 (1), 39-71.

Chieu, H.L. & Ng Hwee Tou (2002). Named entity recognition: a maximum entropy approach using global information. In *COLING 2002. Proceedings of the 19th International conference on Computational Linguistics* (pp. 190-196). San Francisco: Morgan Kaufmann.

Collins, M. & Duffy, N. (2002). Convolution kernels for natural language. In T.G. Dieterich, S. Becker & Z. Ghahramani (Eds.), *Advances in Neural Information Processing Systems* 14 (pp. 625-632. Cambridge, MA: The MIT Press.

Culotto, Aron and Jeffrey Sorenson (2004). Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics* (pp. 424-430). East Stroudsburg, PA: ACL.

Krishnan, V. & C.D. Manning (2006). An effective two-stage model for exploiting non-local dependencies in named entity recognition. *Proceedings of COLING-ACL 2006* (pp. 1121-1128). East Stroudsburg, PA: ACL.

References

Mani, I. et al. (2006). Machine learning of temporal relations. In *Proceedings of COLING-ACL 2006* (pp. 753-760). East Stroudsburg, PA: ACL.

Manning, C.D., Raghaven, P. & Schüze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Moens, M.-F. (2006). *Information Extraction: Algorithms and Prospects in a Retrieval Context* (*The Information Retrieval Series* 21). New York: Springer.

Riloff, E. (1996). An empirical study for automated dictionary construction for information extraction in three domains. *Artificial Intelligence* 85, 101-134.

Soderland, S. (1999). Learning information extraction rules for semi-structured and free text. *Machine Learning:* http://citeseer.nj.nec.com/soderland99learning.html

Sutton, C. & McCallum A. (2007). Introduction to statistical relational learning. In L. Getoor & B. Taskar (Eds.), *Introduction to Statistical Relational Leaning* (pp. 93-127). Cambridge, MA: The MIT Press.

Vishwanathan, S.V.N. & Smola, A.J. (2004). Fast kernels for string and tree matching. In K. Tsuda, B. Schölkopf and J.P. Vert (Eds.), *Kernels and Bioinformatics*. Cambridge, MA, The MIT Press.