Towards a Grand Unified System for Data Mining

Vikram Pudi
IIIT Hyderabad
vikram@iiit.ac.in
http://www.iiit.ac.in/~vikram

Data Mining

= Automated discovery of *interesting patterns* in large datasets

- Researchers identified several kinds of interesting patterns in an adhoc manner
  - classification and regression models, clusters, association rules, frequent patterns, sequential patterns, time-series patterns, summaries, cyclic patterns, hierarchical patterns, max-patterns, closed patterns, multi-dimensional patterns, etc.

Current State-of-Art

- Dozens of algorithms exist for each task, focused on optimizing accuracy, speed, etc.
- State-of-art algorithms are usually mathematically robust and proven to work effectively for *specific domains* (language, speech, image, web, etc.).
- But these algorithms are often conceptually complex, hard to "break into pieces", and hence cannot be easily customized for new domains.

Customizing Difficulty

- Applying current algorithms in real-life requires data-mining experts
  - To map domain problem to data mining tasks
  - To select which algorithms to use for each task
  - To set parameters, select features, design distance metrics
- Unfortunately, it also requires domain experts
- Both kinds of experts are costly!
- Communication between data-mining experts and domain experts is often a bottle-neck.
  - Experts have depth in their respective domains.
  - Not guaranteed to have communication skills good enough to translate specialized terminology.

Unified Theory of Data Mining

- 1 of the 10 recently identified challenging problems [Yang & Wu] is to develop a *unifying theory of data mining.*
  - ⇒ Is there a small set of core data mining tasks to which all others can be reduced?
  - ⇒ It may be possible to build a data mining system that implements the core in a highly flexible, modular way.
  - ⇒ Application of data mining to specific domains (language, speech, image, web, etc.), and to other data mining tasks becomes a matter of *simple customization*.

Challenge: Grand Unified System

- Create a complete data mining system that is easily extensible to unforeseen requirements and new domains.
  - Reduce dependency on data mining experts
    - Extensible and customizable
    - ⇒ Simple to understand, design, implement, modify, break into pieces.
    - Without sacrificing on standard metrics such as accuracy and speed, as far as possible.
  - Reduce dependency on domain experts
    - Minimize user-defined parameters, or set them automatically.
    - Automate feature selection.
    - Automate design of distance metrics.

Desirable Features of System

1. *Simple and modular*
   - Generic: Built on principles that are domain independent
   - Customizable: Allow embedding of domain constraints
   - Data Driven: Better, *discover* domain constraints from data
2. Accurate
3. Handle dynamic data
4. Efficient offline processing (scalable)
5. Efficient online processing (interactive response times)
6. Parameterless
7. Noise-resistant

Idea 1: The Scientific Method

❑ *Observe* system behaviour and collect data.

⇐ *Model* behaviour using rules / theories.

⇐ *Predict* behaviour using models.

⇐ *Design* systems with desired behaviour using our predictive capability.

Idea 1: The Goal of Scientific Knowledge

❑ *Observe* system behaviour and collect data.
- ❑ Repeatable observations ⇒ Frequent patterns
- ❑ Significance of observations ⇒ Uniqueness mining

⇐ *Model* behaviour using rules / theories.
- ❑ Association rules, probability distributions, …

⇐ *Predict* system behaviour using models.
- ❑ Classification, Regression

⇐ *Design* systems with desired behaviour using our predictive capability.
- ❑ Search a space of designs that are similar to known good designs: Similarity search

Idea 2: Frequent Patterns as Model

- That which is *infrequent* is *insignificant*.
  - Although in rare cases it may be significant, there is not enough statistical evidence to conclude anything about it.
  - ⇒ The set of frequent patterns represent everything that is significant in the data.
  - ⇒ Overall trends and patterns that can be inferred from the original dataset can always be inferred from the frequent itemsets.
- The resulting representation of frequent patterns is typically much smaller than the original dataset size. Moreover, we can control this size according to our requirements and capacity.

Idea 2: Frequent Patterns and Features

Input data format for most mining tasks

Outlook	Temp (°F)	Humidity (%)	Windy ?	Class
sunny	75	70	true	*play*
sunny	80	90	true	*don't*
sunny	85	85	false	*don't*
sunny	72	95	false	*don't*
sunny	69	70	false	*play*
overcast	72	90	true	*play*
overcast	83	78	false	*play*
overcast	64	65	true	*play*
overcast	81	75	false	*play*
rain	71	80	true	*don't*
rain	65	70	true	*don't*
rain	75	80	false	*play*
rain	68	80	false	*play*
rain	70	96	false	*play*

⇒

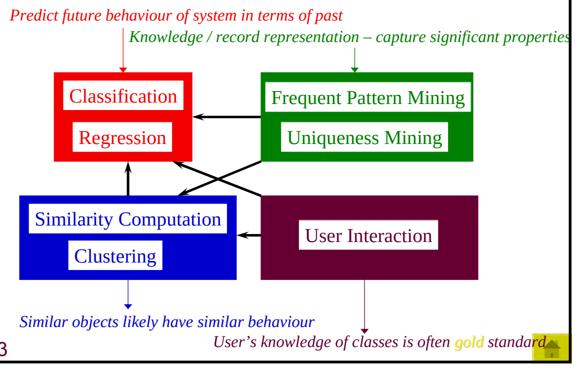
Market-basket format
sunny, t=(70,79), hum=(70,79),windy,*play*
sunny, t=(80,89),hum=(90,99),windy,*don't*
sunny, t=(80,89),hum=(80,89), still, *don't*
sunny, t=(70,79),hum=(90,99), still, *don't*
sunny, t=(60,69),hum=(70,79), still, *play*
overcast, t=(70,79),hum=(90,99),windy, *play*
overcast, t=(80,89),hum=(70,79), still, *play*
overcast, t=(60,69),hum=(60,69),windy, *play*
overcast, t=(80,89),hum=(70,79), still, *play*
rain, t=(70,79),hum=(80,89),windy, *don't*
rain, t=(60,69),hum=(70,79), windy, *don't*
rain, t=(70,79),hum=(80,89),still, *play*
rain, t=(60,69),hum=(80,89), still, *play*
rain, t=(70,79),hum=(90,99), still, *play*

- Frequent patterns mined from market-basket data are indicative of certain trends. E.g. a pattern may be indicative of a specific class.
- We can infer trends / behaviours of individual records by analyzing the frequent patterns present in them.

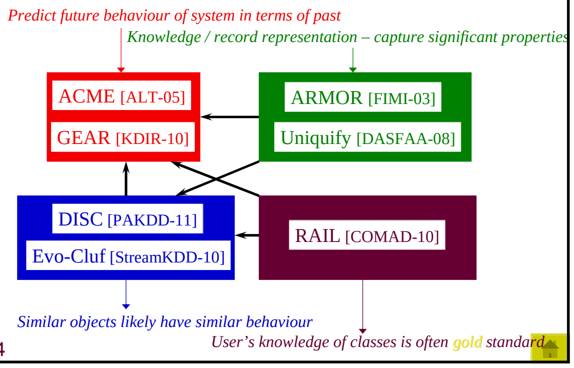
Advantages of Frequent Patterns

- Frequent patterns capture all significant relationships between items in a dataset.
- Ensuring a minimum frequency eliminates *noise*.
- The *size* of the resulting representation is small and controllable.
- *Efficient* frequent pattern mining algorithms exist that easily handle 1000s of columns and billions of rows.
- Efficient *incremental* mining algorithms exist to handle changing datasets.
- For a new domain, we can input a large space (1000s) of possible features without worrying too much about feature selection. The algorithms can figure out which are statistically significant feature combinations.

Core Problems of Data Mining


Predict future behaviour of system in terms of past
Knowledge / record representation – capture significant properties
Classification
Regression
Frequent Pattern Mining
Uniqueness Mining
Similarity Computation
Clustering
User Interaction
Similar objects likely have similar behaviour
User's knowledge of classes is often gold standard

Grand Unified System


Predict future behaviour of system in terms of past
Knowledge / record representation – capture significant properties
ACME [ALT-05]
GEAR [KDIR-10]
ARMOR [FIMI-03]
Uniquify [DASFAA-08]
DISC [PAKDD-11]
Evo-Cluf [StreamKDD-10]
RAIL [COMAD-10]
Similar objects likely have similar behaviour
User's knowledge of classes is often gold standard

Features

	ARMOR	ACME	Uniq.	Evo-Cluf	GEAR	DISC	RAIL
Generic	Y	Y	Y	Y	Y	Y	Y
Data Driven	Y	Y	Y	Y	Y	Y	Y
Accurate	n/a	Y	todo	Y	Y	Y	Y
Dynamic Data	Y	N	can	Y	Y	Y	Y
Scalable	Y	N	Y	Y	Y	Y	Y
Interactive response	can	Y	Y	todo	Y	Y	Y
Parameter -less	N	Y	N	Y	Y	Y	N
Noise Resistant	Y	Y	Y	Y	Y	Y	Y

ARMOR: Mine Association Rules

- Define optimal algorithm (*Oracle*)
  - Magically knows identities of frequent itemsets before mining begins. Has to only determine counts of these itemsets in *one* pass over the database.
- *Minimal* changes to Oracle
- Maximum *two* passes over database
- "Short and light" second pass
- Performance: Within *twice* of Oracle for a variety of real and synthetic databases
- Easy to convert to *incremental* algorithm and to apply on *data streams*.

ACME: Classifier

- The frequent itemsets of each class, with their probabilities are used as *constraints* in a max-entropy model.
  - Max-entropy ⇒ Mathematically robust
  - Frequent itemsets ⇒ all significant constraints
  - Best in theory and practice
- But, slow.

Uniquify: Uniqueness Mining

- To determine what properties makes each given record (object) unique/special.
  - Hiring people that have special talents
  - Assigning jobs based on speciality
  - Assigning weights to special jobs
  - Assigning marks to questions in an exam
- Formulation: A property is unique if there are very few objects with that property, while for *similar* (sibling) properties this is not the case.
  - Example properties: (lang='English'), (country='India',lang='Hindi'),etc.
- Lemma: If a property *p* is unique, every specialization of *p* is also unique.
  - ⇒ Levelwise mining is possible.

EvoCluf: Clustering

- Mine generalized closed frequent itemsets
- Each frequent itemset is a cluster. Hierarchy of clusters exists.
  - Remove duplication by finding cluster with maximum score for a record d.
    - score(d,C) = Sum of TF-IDF of each item of C in d
    - score(d,C) = Sum of similarity of d and each record in C
    - E.g. For text documents, can use no. of matching wikipedia categories of words in the documents
- Evolution: We use incremental algorithm to find a new clustering at time t+1, and update records to belong to their new best clusters.
- Good quality clusters (based on F-score) are obtained, while at the same time ensuring that cluster evolution is smooth (i.e. do not change abruptly).

19

GEAR: kNN-based Regression

- Assume dependent variable varies smoothly.
- Smooth curves can be modelled as piece-wise linear.
  - ⇒ Apply local linear regression.
- Some Details:
  - Find *k* nearest neighbours
    - Select best *k*: Vary *k* and select one which minimizes error
  - Construct predictors: For each dimension, construct a line that fits the *k* nearest neighbours along that dimension.
  - Output: Weighted sum of values output by individual predictors.
    - Set weight inversely proportional to mean error of prediction (of *k*NNs) along the dimension of predictor.
- Performs better than 14 other algorithms (including state-of-art) on standard datasets.

20

DISC: Data Driven Similarity

- Introduce notion of similarity between 2 values a_{ij} and a_{ik} of a categorical attribute A_i based on *co-occurrence statistics* and *interestingness* of co-occurrences.
  - Define: M_{ij} = [*interestingness*($a_{ij} \rightarrow v$): *all values v of all attributes except* A_i]
  - Similarly define M_{ik}
  - Measure vector similarity of M_{ij} and M_{ik}
- Tried 12 interestingness measures and 3 vector similarity functions.
- Evaluated for clustering (k-means) and classification (kNN) tasks
  - Significant improvement in accuracy by changing only similarity measure while keeping algorithm and its parameters constant.

21

RAIL: Interactive Classification

- Manual intervention should be minimal
- Use associative classifier
  - Mine CARs [Consequent is class].
  - To classify test record R, use weighted voting of CARs that match R.
- When human classifies records (whose ambiguity > μ):
  - Change weight of existing CARs
  - Create new *soft* CARs using minimally infrequent (negative border) patterns.
  - Promote soft CARs to hard when they become frequent.
- Achieves high accuracy with very few user pings.

22

Association Rule Mining based on Oracle

ARMOR

In Frequent Itemset Mining Implementations (FIMI) 2003

Frequent Itemsets

$\mathcal{D}$:

Transaction ID	Items
1	Tomato, Potato, Onions
2	Tomato, Potato, Brinjal, Pumpkin
3	Tomato, Potato, Onions, Chilly
4	Lemon, Tamarind

Rule: Tomato, Potato → Onion (confidence: 66%, support: 50%)

Support(X) = |transactions containing X| / |$\mathcal{D}$|

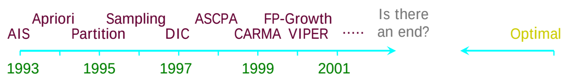
Confidence(R) = support(R) / support(LHS(R))

Problem proposed in [AIS 93]: Find all rules satisfying user given minimum support and minimum confidence.

⇒ Find all *frequent* itemsets (i.e. *support*(X) ≥ *minsup*)

24

Feeding Frenzy

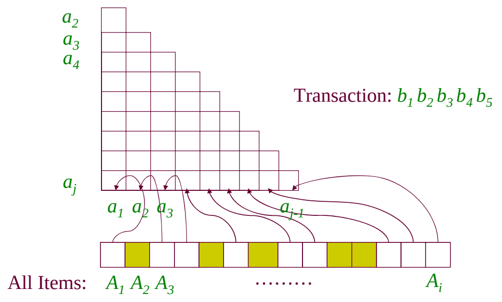


- AIS
- SETM
- Apriori
- AprioriTid
- AprioriHybrid
- OCD
- Partition
- Sampling
- DHP
- CARMA
- ClusterApr
- Eclat
- MaxEclat
- Clique
- MaxClique
- TopDown
- dEclat
- OPUS_AR
- BFPA
- DFPA
- SPINC
- AS-CPA
- SSAS-CPA
- RSAS-CPA
- ColumnWise
- Hierarchical BitMap
- FP-Growth
- Tree Projection
- VIPER
- H-Mine

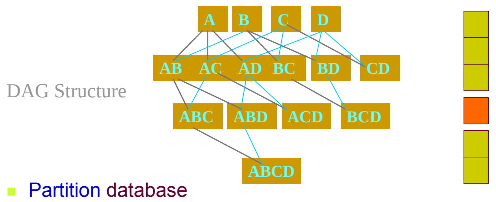
Optimal Algorithm: Oracle

- Magically knows identities of frequent itemsets before mining begins. Therefore, has to only determine the counts of these itemsets in *one* pass over the database
- Minimum work required from any algorithm
- Careful design of data structures to ensure optimal access and enumeration of itemsets

Counting 1&2-itemsets



Counting Longer Itemsets (k >2)

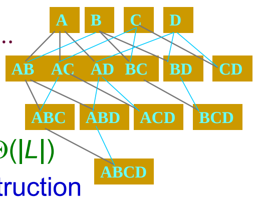


- Partition database
- For each itemset, compute the list of transaction-ids (*tidlist*) containing it
- Initiate tidlist intersections from frequent singletons
- Depth-first traversal
- Optimize using tid-vector approach

Tidset Intersection

Tid-vector V: 0010101101….

Tid-list L: 5, 6, 10, 29, …



- Cost of intersection = $\Theta(|L|)$
- Cost of tid-vector construction
  - Proportional to number of “1”s in V
  - Amortized over many intersections
  - Space for V can be statically allocated

No wasted Enumeration

- All 1-itemsets are either frequent or in -ve border
- Only combinations of *frequent* 1-itemsets enumerated for pairs
- Depth-first search ensures each itemset is visited only *once*

Enumeration Cost = Θ(1)

- Direct lookup arrays for 1&2-itemsets. Best in unit-cost RAM model
- For longer itemsets, cost = Θ(|*X.childset*|) resulting in Θ(1) cost per itemset overall
- All operations involve array and pointer lookups, which cannot be improved upon

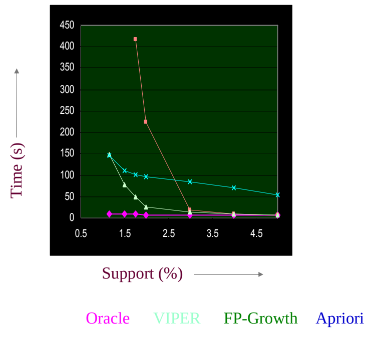
Oracle Features

- Uses direct lookup arrays for 1-itemsets and 2-itemsets
- Uses DAG structure for longer itemsets
- No wasted enumeration of itemsets
- Enumeration cost per itemset = Θ(1)
- Caveat: Not really optimal
  - Doesn't share work for transactions that are significantly similar. E.g. if 2 transactions are identical, it does the same work for both

Performance Setup

- Algorithms: Oracle, VIPER, FP-growth, Apriori
- Variety of Databases
  - File-system backend
  - Integration with commercial RDBMS
    - Cache data to file-system and run algorithm
    - Implement algorithm as stored procedure
    - Implement algorithm in SQL
- Extreme and typical values of *minsup*

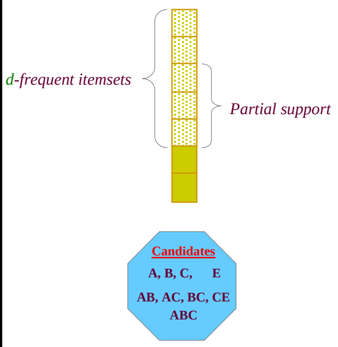
Response Times of Current Algorithms



ARMOR

- *Minimal* changes to Oracle
- Maximum *two* passes over database
- "Short and light" second pass
- Performance: Within *twice* of Oracle for a variety of real and synthetic databases

ARMOR Processing



First Pass

- Conceptually *partition* database into disjoint blocks.
- *C* (the set of *candidates*) = set of all 1-itemsets
- After processing each partition, *update C*; this may involve both insertions and deletions
- *d-frequent itemsets*: The algorithm ensures that at any stage, if *d* is the database scanned so far, then the frequent itemsets within *d* are available.
- *Partial Supports*: The count of an itemset *since* it has been inserted into *C*.

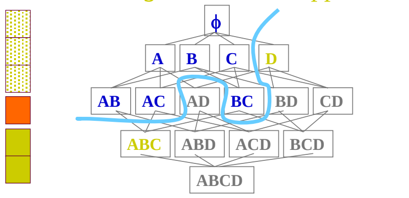
Second Pass

- Obtain *complete supports* of candidates.
- No new insertions into *C*; but deletions allowed.
- Light and short pass

Candidate Generation

Itemsets can move freely between being partially-frequent, negative border and partially-infrequent.

The Negative Border Approach

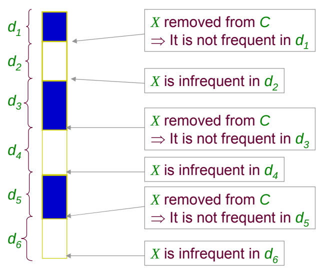


Observation: An itemset can become partially frequent iff it has some subset in *N* which moves to *F*. Such itemsets are called *promoted borders.*

37

Proof of Correctness

Consider the life of an itemset *X* in the set of candidates, *C*
Solid area represents that *X* was in *C*
Blank area represents that *X* was *not* in *C*

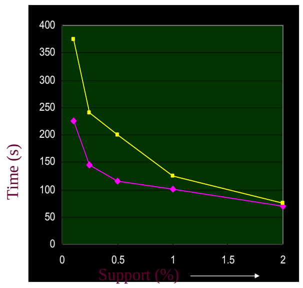


d_1
d_2
d_3
d_4
d_5
d_6
X removed from C $\Rightarrow$ It is not frequent in d_1
X is infrequent in d_2
X removed from C $\Rightarrow$ It is not frequent in d_3
X is infrequent in d_4
X removed from C $\Rightarrow$ It is not frequent in d_5
X is infrequent in d_6

Since *X* is infrequent in every block, it is infrequent overall.

38

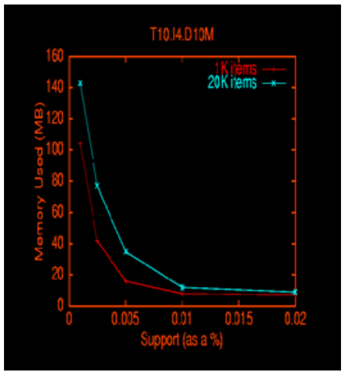
Response Times of ARMOR



Oracle Armor

39

Memory Utilization of ARMOR



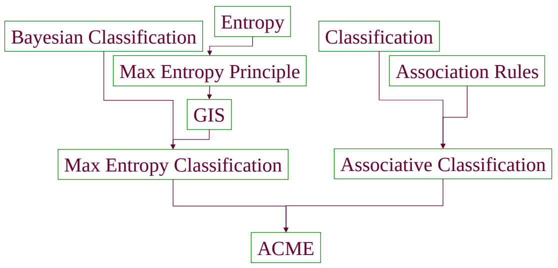
40

Associative Classification based on Maximum Entropy

ACME

In Algorithmic Learning Theory (ALT) 2005

Concepts



42

Classification Problem

Outlook	Temp (°F)	Humidity (%)	Windy?	Class
sunny	75	70	true	*play*
sunny	80	90	true	*don't play*
sunny	85	85	false	*don't play*
sunny	72	95	false	*don't play*
sunny	69	70	false	*play*
overcast	72	90	true	*play*
overcast	83	78	false	*Play*
overcast	64	65	true	*Play*
overcast	81	75	false	*play*
rain	71	80	true	*don't play*
rain	65	70	true	*don't play*
rain	75	80	false	*play*
rain	68	80	false	*play*
rain	70	96	false	*play*

sunny	77	69	true	?
rain	73	76	false	?

Play Outside?

Model relationship between class labels and attributes

e.g. outlook = overcast ⇒ class = *play*

⇒ Assign class labels to new data with *unknown* labels

Recap: Frequent Itemsets

$\mathcal{D}$:

Transaction ID	Items
1	Tomato, Potato, Onions
2	Tomato, Potato, Brinjal, Pumpkin
3	Tomato, Potato, Onions, Chilly
4	Lemon, Tamarind

Rule: Tomato, Potato → Onion (confidence: 66%, support: 50%)

Support(X) = |transactions containing X| / |$\mathcal{D}$|

Confidence(R) = support(R) / support(LHS(R))

Problem proposed in [AIS 93]: Find all rules satisfying user given minimum support and minimum confidence.

⇒ Find all *frequent* itemsets (i.e. *support*(X) ≥ *minsup*)

Associative Classifiers: CPAR, CMAR, …

- Separate training data for each class
- Find frequent itemsets in each class
  - Class Association Rules: LHS = frequent itemset, RHS = class label
- To classify record *R*, find rules that apply on *R*.
- Combine the evidence of rules to decide which class *R* belongs to.
  - E.g. Add the probabilities of the best *k* rules.
  - Mathematically incorrect, but works well in practice.

Shannon's Entropy

- An expt has some possible outcomes
- Consider a series of N expts
- Suppose each outcome occurs exactly M times
  - ⇒ There are N/M possible outcomes
  - ⇒ To represent each outcome, we need log N/M bits.
- This generalizes even when all outcomes are not equally frequent.
  - Reason: For an outcome j that occurs M times, there are N/M equi-probable (pseudo) events among which only one cp to j
- Since p_i = M / N, information content of an outcome is -log p_i. So, expected info content: $H = - \Sigma\, p_i \log p_i$

$$\underbrace{\boxed{M}\boxed{M}\boxed{M}\boxed{M}\boxed{M}\boxed{M}\boxed{M}\boxed{M}}_{\text{N/M outcomes}} \quad \text{Total = N}$$

Maximum Entropy Principle

- Entropy corresponds to the disorder (randomness) in a system
  - Intuition: A highly ordered system will require less bits to represent it
  - Uniform random distribution has highest entropy
  - Order ⇔ Constraints [equations, inequations]
- No evidence for order = No order!
- No order = more entropy
- Hence maximize entropy
  - Satisfy known constraints, keep everything else as uniform as possible
- If the constraints are consistent, there is a unique solution that maximizes entropy.

Example

- Constraint 1: Person P distributes Rs.100 to persons A,B,C,D,E.
- If you are forced to guess, how much P gives to each person, what would you guess?

Example

- Constraint 1: Person P distributes Rs.100 to persons A,B,C,D,E.
- Constraint 2: Person P gives Rs.40 to person A.

How much does P give to the other people?

49

Example

- Constraint 1: Person P distributes Rs.100 to persons A,B,C,D,E.
- Constraint 2: Person P gives Rs.40 to person A.

How much does P give to the other people?

We like the most uniform distribution that satisfies known constraints.

50

Log-Linear Modeling

Theorem: If there exists a positive probability distribution of the following form satisfying known constraints, then it maximizes entropy. [GIS; 1972]

$$P(X_k) = \mu_0 \prod_{i \in \text{Constraints}} \mu_i^{f_i(x)}$$

$$\begin{aligned} f_i(X_k) &= 1, \quad \text{if } X_k \text{ satisfies constraint } i \\ &= 0, \quad \text{otherwise} \end{aligned}$$

μ_0 is to ensure that $\sum_k P(X_k) = 1$

These μ's can be computed by an iterative fitting algorithm like the GIS algorithm.

51

Generalized Iterative Scaling (GIS)

```
# N items, M constraints
$P(X_k) = 1 / 2^N$ // for k = (1…$2^N$); Uniform distribution
$\mu_j = 1$ # for j = (1…M)
while all constraints *not* satisfied:
  for each constraint $C_j$:
    $S_j = \Sigma_{(k:\ X_k \text{ satisfies } Y_j)} P(X_k)$
    $\mu_j$ *= $d_j / S_j$
    $P(X_k) = \mu_0 \Pi_{(j \text{ satisfied by } X_k)} \mu_j$
```

52

Bayesian Classification

- Think emails, keywords, spam / non-spam
- Given a new data point $X=\{x_1,x_2,\ldots,x_m\}$ to classify calculate $P(C_i/X)$ for each class C_i.
- Select C_i for which $P(C_i/X)$ is maximum

$$\begin{aligned} P(C_i/X) &= P(X/C_i)\, P(C_i) / P(X) \\ &\propto P(X/C_i)\, P(C_i) \end{aligned}$$

- Naïve Bayes assumes that each x_i is independent
- Instead estimate $P(X/C_i)$ directly from training data: *support*$_{C_i}(X)$
- Problem: There may be no instance of X in training data.
  - Training data is usually sparse

53

Max Entropy in Classification

To predict $P(X/C_i)$:

- Form constraints and solve for $P(X/C_i)$
  - Use *domain expertise* to form constraints
- Among possible solutions, choose the one that has maximum entropy

54

No Free Lunch

- With no assumptions about the domain, is there a *best classification method*?
- Is any algorithm better than random guessing?
- Answer is No!

55

Why No Free Lunch ?

	x	F	h1	h2
	000	1	1	1
D	001	2	2	2
	010	1	1	1
	011	2	1	2
	100	1	1	2
	101	2	1	2
	110	1	1	2
	111	1	1	2

h1: Always selects class 1
h2: Always selects class 2
Off-training set error:
E1=0.4, E2=0.6
h1 is better.

Averaged over all target functions there is no difference in off-training set errors.

56

Is there a best classifier?

- Consider any classifier that classifies a given record *R* into class 1. The number of target functions for which this is correct it is the same as the number for which it is wrong.
- "Classifier 1 is better than classifier 2" are ultimately statements about the underlying target functions.

57

Interpretation of Optimality

- There is no best classifier, unless we make some assumptions about the distribution of unseen data.
- Similar Familiar Problem: There is no best sorting algorithm, unless we make some assumptions about the data distribution.
  - Yet we compare algorithms using order-complexity.
- Caveat: Sorting algorithms can make one pass to determine the data distribution. Classifiers cannot *see unseen data.*

58

Max Entropy vs No Free Lunch

- No Free Lunch: With no prior information, assume every target function equally likely.
  - $\Rightarrow$ Every classifier is equally good.
- Max Entropy: If we have prior information, then use it to determine the distribution of target function.
  - Target functions that satisfy known constraints are preferred.
  - Among such target functions, none is preferred over the other.
  - Max entropy classifier will perform best when averaged over target functions that satisfy the known constraints.

59

ACME

- The frequent itemsets of each class, with their probabilities are used as *constraints* in a max-entropy model.
  - Max-entropy $\Rightarrow$ Mathematically robust
  - Frequent itemsets $\Rightarrow$ all significant constraints
  - Best in theory and practice
- But, slow.

60

Split Data by Classes

C_1

Records
a, b, c
b, c
a, c

C_2

Records
a, d

Apriori

Apriori

$\mathbf{S_1}$

$\mathbf{S_2}$

Frequent Itemsets of C_1

Frequent Itemsets of C_2

$$S = S_1 \cup S_2$$

Build Constraints for a Class

Constraints of $C_i = \{ (s_j, p_j) \mid s_j \in S \wedge P(s_j \mid C_i) = p_j \}$

C_1

Records
a, b, c
b, c
a, c

$s_j \in S$

Constraints of C_1

s_j	p_j
b, c	0.67
a, b, c	0.33
b	0.67

Build distribution of class C_1

constraints

s_i	p_i
b, c	0.67
a, b, c	0.33
b	0.67

X				
a	b	c	d	$P(X\|C_1)$
1	1	1	1	0.1
1	1	1	0	0.1

Total possible records – 2^4 in number

Maximum Entropy Principle: Build a distribution $P(X|C_1)$ that conforms to the constraints and has the highest *Entropy*

Problem with Log-Linear Model

Model: $P(X \mid C_j) = \prod_{s_i \in C_j} \mu_i^{f_i(x)}$

Solution does not exist if $P(X|C_j) = 0$ for any X.

s_i	p_i
b, c	0.67
a, b, c	0.33
b	0.67

$b \Rightarrow c$

Probability is 0 for all ‘X’ which have ‘b=1’ but ‘c=0’.

Fix to the Model

Fix: Define the model on only those ‘X’ whose probability is non-zero.

Explicitly set these record probabilities to zero and learn for μ’s without considering them.

X				
a	b	c	d	$P(X\|C_1)$
0	1	0	0	set to 0
0	1	0	1	set to 0
1	1	0	0	set to 0
1	1	0	0	set to 0

Learning time decreases as |X| decreases

Effect of pruning

- Datasets chosen from UCI ML Repository.

Dataset	# Cons	Pruned X
Austra	(354) 263	10.1%
Waveform	(99) 24	1.3%
Cleve	(246) 204	9.96%
Diabetes	(85) 61	7.42%
German	(54) 44	9.66%
Heart	(115) 95	55.1%
Breast	(189)180	8.89%
Lymph	(29) 14	12.1%
Pima	(87) 55	8.6%

Making the approach scalable (1)

- Remove non-informative constraints.
  - A constraint is informative if it can distinguish between classes very well. Use standard information measure
  - Entropy of the distribution $P(C|s_i)$ should be greater than a given threshold (0.6 in our experiments).

Eg: $s_1 = \{a,b,c\}$

$P(C_1|s_1) = 0.45$ and $P(C_2|s_1) = 0.55$

Remove {a,b,c} from the constraint set.

$s_2 = \{b, c\}$

$P(C_1|s_2) = 0.8$ and $P(C_2|s_2) = 0.2$

Include { b, c } in the constraint set.

Making the approach scalable (2)

- Splitting: Split the set of features 'I' into groups that are independent of each other.
  - Two groups of features are independent of each other if they don't have an overlapping constraint between them
- Global P(.) can be calculated by merging individual P(.)'s of each group in a naïve-bayes fashion

Ex: I = {a,b,c,d}, and constraints are {a}, {a,b} and {c,d}. Split I into I_1={a,b} and I_2={c,d}.

Learn Log-Linear models $P_1(.)$ for I_1={a,b} and $P_2(.)$ for I_2={c,d}

$P(b,c) = P_1(b) * P_2(c)$

Performance Evaluation

Performance of ACME vs other classifiers.

Dataset	NB	C4.5	CBA	TAN	ACME
Austra	84.34	85.5	84.9	84.9	**85.5**
Breast	97.28	95.13	96.3	96.56	**96.49**
Cleve	83.82	76.23	82.8	82.5	**83.82**
Diabetes	75.78	72.39	74.5	76.30	**77.86**
German	70.0	70.9	73.4	73	**71.3**
Heart	83.7	80	81.87	81.85	**82.96**
Lymph	83.1	77.0	77.8	84.45	**78.4**
Pima	76.17	74.34	72.9	76.3	**77.89**
Waveform	80.82	76.44	80.0	81.52	**83.08**

Example

- Imagine emails represented as sets of keywords and being classified as spam/non-spam.
- Let us focus on 3 keywords A, B, C.
- From the training data, let us say we get 4 "constraints" for the spam class:
  - P(A) = 0.2 = d_1 (say)
  - P(B) = 0.3 = d_2
  - P(C) = 0.1 = d_3
  - P(AB) = 0.1 = d_4
- Now, our task is to use the GIS algorithm to determine P(ABC).

- Define X_i as a bit-vector representing the presence/absence of keywords.
- E.g. X_3 = 011 represents the presence of B,C and absence of A.
- Note that P(X_3) is "not" the same as P(BC) since the latter doesn't care about the presence/absence of A.

```
      ABC
X_0 = 000
X_1 = 001
X_2 = 010
X_3 = 011
X_4 = 100
X_5 = 101
X_6 = 110
X_7 = 111
```

The previous 4 constraints can be rewritten in terms of the X_i's:

P(A) = P(X_4) + P(X_5) + P(X_6) + P(X_7) = 0.2
P(B) = ... = 0.3
P(C) = ... = 0.1
P(AB) = P(X_6) + P(X_7) = 0.1
//AB is satisfied by X_6 and X_7

Define:
S_1 = P(X_4) + P(X_5) + P(X_6) + P(X_7) ----(1)
S_2 = ... ---(2)
S_3 = ... ---(3)
S_4 = P(X_6) + P(X_7) -----------------------------(4)

Start GIS with P(X_i) = 1/8 = 0.125 (for all i)
Also, set mu_1 = mu_2 = mu_3 = mu_4 = 1 and mu_0 = 1
There are 4 mu's because there are 4 constraints.

Next, we calculate S_i for each constraint i, as per (1), (2), (3), (4) above.
S_1 = 1/8 + 1/8 + 1/8 + 1/8 = 0.5
similarly, S_2 = S_3 = 0.5, S_4 = 0.25

S_1 is 0.5; but we want it to be 0.2 (since P(A) = 0.2). Since S_1 is the sum of some P(X_i)'s in equation (1), we want to reduce these P(X_i)'s. We want to scale them down by 0.2 / 0.5. So we set: mu_i *= d_i / S_i. Thus, we get:

mu_1 = 1 * 0.2 / 0.5 = 0.4
mu_2 = 1 * 0.3 / 0.5 = 0.6
mu_3 = 1 * 0.1 / 0.5 = 0.2
mu_4 = 1 * 0.1 / 0.25 = 0.4

Using these mu's, we recalculate P(X_i)'s as:
P(X_i) = product of those mu_j's whose cp constraint is satisfied by X_i
Thus:
P(X_0) = 1   // X_0 (000) doesn't satisfy any constraint
P(X_1) = mu_3   // X_1 (001) satisfies constraint 3 only
P(X_2) = mu_2
P(X_3) = mu_2 * mu_3 // X_3 (011) satisfies constraints 2 & 3
P(X_4) = mu_1
P(X_5) = mu_1 * mu_3
P(X_6) = mu_1 * mu_2 * mu_4 // X_6 (110) satisfies constraints 1, 2 & 4
P(X_7) = mu_1 * mu_2 * mu_3 * mu_4 // X_7 (111) satisfies all 4 constraints

But sum of above P(X_i)'s might not turn out to be 1. Infact, it turns out to be 2.5152 for this example. So we scale all of them down by 2.5152 to make the sum equal 1. Then, we get:

P(X_0) = 1 / 2.5152 = 0.4 (approx)
P(X_1) = 0.08
P(X_2) = 0.24
P(X_3) = 0.048
P(X_4) = 0.16
P(X_5) = 0.032
P(X_6) = 0.04
P(X_7) = 0.008

That was the 1st iteration of GIS. These numbers are closer to the actual P(X_i) values. For example, we know that P(A)=0.2, P(B)=0.3, P(C)=0.1. If A,B,C are mutually exclusive, then P(A or B or C) = 0.6 (the sum). Notice that P(X_0) above is 0.4 which is 1 - 0.6. If we run the algorithm for more iterations we get better results.

Our task was to determine P(ABC). So we just output the value of P(X_7) above.