Unsupervised Feature Construction for Improving Data Representation and Semantics

Marian-Andrei Rizoiu, Julien Velcin, Stéphane Lallich

Unsupervised Feature Construction for Improving Data Representation and Semantics

Marian-Andrei Rizoiu · **Julien Velcin** ·
Stéphane Lallich

Abstract Feature-based format is the main data representation format used by machine learning algorithms. When the features do not properly describe the initial data, performance starts to degrade. Some algorithms address this problem by internally changing the representation space, but the newly-constructed features are rarely comprehensible. We seek to construct, in an unsupervised way, new features that are more appropriate for describing a given dataset and, at the same time, comprehensible for a human user. We propose two algorithms that construct the new features as conjunctions of the initial primitive features or their negations. The generated feature sets have reduced correlations between features and succeed in catching some of the hidden relations between individuals in a dataset. For example, a feature like *sky* ∧ ¬*building* ∧ *panorama* would be true for non-urban images and is more informative than simple features expressing the presence or the absence of an object. The notion of Pareto optimality is used to evaluate feature sets and to obtain a balance between total correlation and the complexity of the resulted feature set. Statistical hypothesis testing is used in order to automatically determine the values of the parameters used for constructing a data-dependent feature set. We experimentally show that our approaches achieve the construction of informative feature sets for multiple datasets.

Marian-Andrei Rizoiu · Julien Velcin · Stéphane Lallich
ERIC Laboratory, University Lumière Lyon 2
5, avenue Pierre Mendès France, 69676 Bron Cedex, France
Tel. +33 (0)4 78 77 31 54
Fax. +33 (0)4 78 77 23 75

Marian-Andrei Rizoiu
E-mail: Marian-Andrei.Rizoiu@univ-lyon2.fr

Julien Velcin
E-mail: Julien.Velcin@univ-lyon2.fr

Stéphane Lallich
E-mail: Stephane.Lallich@univ-lyon2.fr

1 Introduction

Most machine learning algorithms use a representation space based on a *feature-based* format. This format is a simple way to describe an instance as a measurement vector on a set of predefined features. In the case of supervised learning, a class label is also available. One limitation of the feature-based format is that supplied features sometimes do not adequately describe, in terms of classification, the semantics of the dataset. This happens, for example, when general-purpose features are used to describe a collection that contains certain relations between individuals.

In order to obtain good results in classification tasks, many algorithms and preprocessing techniques (e.g., SVM (Cortes and Vapnik, 1995), PCA (Dunteman, 1989) *etc.*) deal with non-adequate variables by internally changing the description space. The main drawback of these approaches is that they function as a black box, where the new representation space is either hidden (for SVM) or completely synthetic and incomprehensible to human readers (PCA).

The purpose of our work is to construct a new feature set that is more descriptive for both supervised and unsupervised classification tasks. In the same way that **frequent itemsets** (Piatetsky-Shapiro, 1991) help users to understand the patterns in transactions, our goal with the new features is to help understand relations between individuals of datasets. Therefore, the new features should be easily comprehensible by a human reader. Literature proposes algorithms that construct features based on the original user-supplied features (called primitives). However, to our knowledge, all of these algorithms construct the feature set in a supervised way, based on the class information, supplied *a priori* with the data.

In order to construct new features, we propose two algorithms that create new feature sets in the absence of classified examples, in an unsupervised manner. The first algorithm is an adaptation of an established supervised algorithm, making it unsupervised. For the second algorithm, we have developed a completely new heuristic that selects, at each iteration, pairs of highly correlated features and replaces them with conjunctions of literals that do not co-occur. Therefore, the overall redundancy of the feature set is reduced. Later iterations create more complex Boolean formulas, which can contain negations (meaning absence of features). We use statistical considerations (hypothesis testing) to automatically determine the value of parameters depending on the dataset, and a *Pareto front* (Sawaragi et al, 1985)-inspired method for the evaluation. The main advantage of the proposed methods over PCA or the kernel of the SVM is that the newly-created features are comprehensible to human readers (features like $people \wedge manifestation \wedge urban$ and $people \wedge \neg urban \wedge forest$ are easily interpretable).

In Sections 2 and 3, we present our proposed algorithms and in Section 4 we describe the evaluation metrics and the complexity measures. In Section 5, we perform a set of initial experiments and outline some of the inconveniences of the algorithms. In Section 6, by use of statistical hypothesis testing, we address these weak points, notably the choice of the threshold parameter. In Section 7, a second set of experiments validates the proposed improvements. Finally, Section 8 draws the conclusion and outlines future works.

1.1 Motivation: why construct a new feature set

In the context of classification (supervised or unsupervised), a useful feature needs to portray new information. A feature p_j, that is highly correlated with another feature p_i, does

Fig. 1: Example of images tagged with $\{groups, road, building, interior\}$

not bring any new information, since the value of p_j can be deduced from that of p_i. Subsequently, one could filter out "irrelevant" features before applying the classification algorithm. But by simply removing certain features, one runs the risk of losing important information of the **hidden structure of the feature set**, and this is the reason why we perform **feature construction**. Feature construction attempts to increase the expressive power of the original features by discovering missing information about relationships between features.

We deal primarily with datasets described with **Boolean** features. Any dataset described by using the feature-value format can be converted to a binary format using discretization and binarization. In real-life datasets, most binary features have specific meanings. Let us consider the example of a set of images that are tagged using Boolean features. Each feature marks the presence (**true**) or the absence (**false**) of a certain object in the image. These objects could include: *water*, *cascade*, *manifestation*, *urban*, *groups* or *interior*. In this case, part of the semantic structure of the feature set can be guessed quite easily. Relations like "is-a" and "part-of" are fairly intuitive: *cascade* is a sort of *water*, *paw* is part of *animal* etc. But other relations might be induced by the semantics of the dataset (images in our example). *manifestation* will co-occur with *urban*, for they usually take place in the city. Fig. 1 depicts a simple image dataset described using the feature set $\{groups, road, building, interior\}$. The feature set is quite redundant and some of the features are non-informative (e.g., feature *groups* is present for all individuals). Considering co-occurrences between features, we could create the more eloquent features $groups \wedge \neg road \wedge interior$ (describing the top row) and $groups \wedge road \wedge building$ (describing the bottom row).

The idea is to create a data-dependent feature set, so that the new features are as independent as possible, limiting co-occurrences between the new features. At the same time they should be comprehensible to the human reader.

1.2 Related work

The literature proposes methods for augmenting the descriptive power of features. Liu and Motoda (1998) collects some of them and divides them into three categories: feature selection, feature extraction and feature construction.

Feature selection (Lallich and Rakotomalala, 2000; Mo and Huang, 2011) seeks to filter the original feature set in order to remove redundant features. This results in a representation space of lower dimensionality. **Feature extraction** is a process that extracts a set of new features from the original features through functional mapping (Motoda and Liu, 2002). For example, the **SVM algorithm** (Cortes and Vapnik, 1995) constructs a kernel function that changes the description space into a new separable one. Supervised and non-supervised algorithms can be boosted by pre-processing with **principal component analysis** (PCA) (Dunteman, 1989). PCA is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of uncorrelated variables, called *principal components*. **Manifold learning** (Huo et al, 2005) can be seen as a classification approach where the representation space is changed internally in order to boost the performances. Feature extraction mainly seeks to reduce the description space and redundancy between features. Newly-created features are rarely comprehensible and very difficult to interpret. Both feature selection and feature extraction are inadequate for detecting relations between the original features.

Feature Construction is a process that discovers missing information about the relationships between features and augments the space of features by inferring or creating additional features (Motoda and Liu, 2002). This usually results in a representation space with a larger dimension than the original space. Constructive induction (Michalski, 1983) is a process of constructing new features using two intertwined searches (Bloedorn and Michalski, 1998): one in the representation space (modifying the feature set) and another in the hypothesis space (using classical learning methods). The actual feature construction is done using a set of constructing operators and the resulted features are often conjunctions of primitives, therefore easily comprehensible to a human reader. Feature construction has mainly been used with decision tree learning. New features served as hypotheses and were used as discriminators in decision trees. Supervised feature construction can also be applied in other domains, like decision rule learning (Zheng, 1995).

Algorithm 1 General feature construction schema.

Input: P – set of primitive user-given features
Input: I – the data expressed using P which will be used to construct features
   Inner parameters: Op – set of operators for constructing features, M – machine learning algorithm to be employed
Output: F – set of new (constructed and/or primitives) features.
  $F \leftarrow P$
  $iter \leftarrow 0$
  repeat
    $iter \leftarrow iter + 1$
    $I_{iter} \leftarrow$ **convert**(I_{iter-1}, F)
    $output \leftarrow$ Run $\mathbf{M}(I_{iter}, F)$
    $F \leftarrow F \bigcup$ new feat. constructed with $\mathbf{Op}(F, output)$
    prune useless features in F
  until stopping criteria are met.

Algorithm 1, presented in Gomez and Morales (2002); Yang et al (1991), represents the general schema followed by most constructive induction algorithms. The general idea is to start from I, the dataset described with the set of primitive features. Using a set of constructors and the results of a machine learning algorithm, the algorithm constructs new features that are added to the feature set. In the end, useless features are pruned. These

steps are iterated until some stopping criterion is met (e.g., a maximum number of iterations performed or a maximum number of created features).

Most constructive induction systems construct features as conjunctions or disjunctions of literals. Literals are the features or their negations. E.g., for the feature set $\{a, b\}$ the literal set is $\{a, \neg a, b, \neg b\}$. Operator sets $\{AND, Negation\}$ and $\{OR, Negation\}$ are both complete sets for the Boolean space. Any Boolean function can be created using only operators from one set. FRINGE (Pagallo and Haussler, 1990) creates new features using a decision tree that it builds at each iteration. New features are conjunctions of the last two nodes in each positive path (a positive path connects the root with a leaf having the class label **true**). The newly-created features are added to the feature set and then used in the next iteration to construct the decision tree. This first algorithm of feature construction was initially designed to solve replication problems in decision trees.

Other algorithms have further improved this approach. CITRE (Matheus, 1990) adds other search strategies like *root* (selects first two nodes in a positive path) or *root-fringe* (selects the first and last node in the path). It also introduces domain-knowledge by applying filters to prune the constructed features. CAT (Zheng, 1998) is another example of a hypothesis-driven constructive algorithm similar to FRINGE. It also constructs conjunctive features based on the output of decision trees. It uses a dynamic-path based approach (the conditions used to generate new features are chosen dynamically) and it includes a pruning technique.

There are alternative representations, other than conjunctive and disjunctive. The M-of-N and X-of-N representations use feature-value pairs. An feature-value pair $AV_k(A_i = V_{ij})$ is **true** for an instance if and only if the feature A_i has the value V_{ij} for that instance. The difference between M-of-N and X-of-N is that, while the second one counts the number of true feature-value pairs, the first one uses a threshold parameter to assign a value of truth for the entire representation. The algorithm $ID2 - of - 3$ (Murphy and Pazzani, 1991) uses M-of-N representations for the newly-created features. It has a specialization and a generalization construction operator and it does not need to construct a new decision tree at each step, but instead integrates the feature construction in the decision tree construction. The $XofN$ (Zheng, 1995) algorithm functions similarly, except that it uses the X-of-N representation. It also takes into account the complexity of the features generated.

Comparative studies like Zheng (1996) show that conjunctive and disjunctive representations have very similar performances in terms of prediction accuracy and theoretical complexity. M-of-N, while more complex, has a stronger representation power than the two before. The X-of-N representation has the strongest representation power, but the same studies show that it suffers from data fragmenting more than the other three.

The problem with all of these algorithms is that they all work in a supervised environment and they cannot function without a class label. In the following sections, we will propose two approaches towards unsupervised feature construction.

2 uFRINGE - adapting FRINGE for unsupervised learning

We propose **uFRINGE**, an unsupervised version of FRINGE, one of the first feature construction algorithms. FRINGE (Pagallo and Haussler, 1990) is a framework algorithm (see Section 1.2), following the same general schema shown in Algorithm 1. It creates new features using a logical decision tree, created using a traditional algorithm like ID3 (Quinlan, 1986) or C4.5 (Quinlan, 1993). Taking a closer look at FRINGE, one would observe that its only component that is supervised is the decision tree construction. The actual construction

of features is independent of the existence of a class attribute. Hence, using an unsupervised decision tree construction algorithm renders FRINGE unsupervised.

Clustering trees (Blockeel et al, 1998) were introduced as generalized logical decision trees. They are constructed using a top-down strategy. At each step, the cluster under a node is split into two, seeking to maximize the intra-cluster variance. The authors argue that supervised indicators, used in traditional decision trees algorithms, are special cases of intra-cluster variance, as they measure intra-cluster **class** diversity. Following this interpretation, clustering trees can be considered generalizations of decision trees and are suitable candidates for replacing ID3 in **uFRINGE**.

Adapting FRINGE to use clustering trees is straightforward: it is enough to replace **M** in Algorithm 1 with the clustering trees algorithm. At each step, uFRINGE constructs a clustering tree using the dataset and the current feature set. Just like in FRINGE, new features are created using the conditions under the last two nodes in each path connecting the root to a leaf. FRINGE constructs new features starting only from positive leaves (leaves labelled true). But unlike decision trees, in classification trees the leaves are not labelled using class features. Therefore, uFRINGE constructs new features based on all paths from root to a leaf.

Newly-constructed features are added to the feature set and used in the next classification tree construction. The algorithm stops when either no more features can be constructed from the clustering tree or when a maximum allowed number of features have already been constructed.

Limitations. uFRINGE is capable of constructing new features in an unsupervised context. It is also relatively simple to understand and implement, as it is based on the same framework as FRINGE. However, it suffers from a couple of drawbacks. Constructed features tend to be redundant and contain doubles. Newly-constructed features are added to the feature set and are used, alongside old features, in later iterations. Older features are never removed from the feature set and they can be combined multiple times, thus resulting in doubles in the constructed feature set. What is more, old features can be combined with new features in which they already participated, therefore constructing redundant features (e.g., f_2 and $f_1 \wedge f_2 \wedge f_3$ resulting in $f_2 \wedge f_1 \wedge f_2 \wedge f_3$). Another limitation is controlling the number of constructed features. The algorithm stops when a maximum number of features are constructed. This is very inconvenient, as the dimension of the new feature set cannot be known in advance and is highly dependent on the dataset. Furthermore, constructing too many features leads to overfitting and an overly complex feature set. These shortcomings could be corrected by refining the constructing operator and by introducing a filter operator.

3 uFC - a greedy heuristic

We address the limitations of uFRINGE by proposing a second, innovative approach. We propose an iterative algorithm that reduces the overall correlation of features of a dataset by iteratively replacing pairs of highly correlated features with conjunctions of literals. We use a greedy search strategy to identify the features that are highly correlated, then use a construction operator to create new features. From two correlated features f_i and f_j we create three new features: $f_i \wedge f_j$, $f_i \wedge \overline{f_j}$ and $\overline{f_i} \wedge f_j$. In the end, both f_i and f_j are removed from the feature set. The algorithm stops when no more new features are created or when it has performed a maximum number of iterations. The formalization and the different key
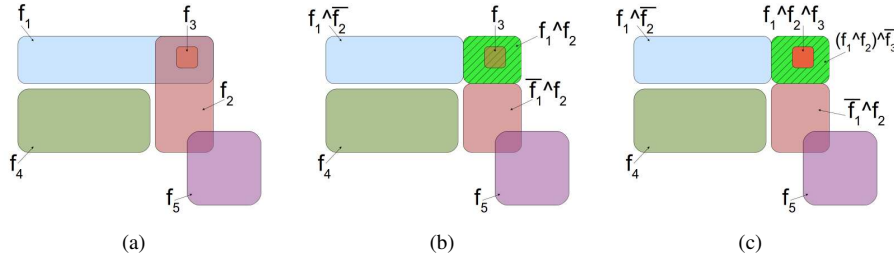
Fig. 2: Graphical representation of how new features are constructed - Venn diagrams. (a) Iter. 0: Initial features (Primitives), (b) Iter. 1: Combining f_1 and f_2 and (c) Iter. 2: Combining $f_1 \wedge f_2$ and f_3

parts of the algorithm (e.g., the search strategy,construction operators or feature pruning) will be presented in the next sections.

Fig. 2 illustrates visually, using Venn diagrams, how the algorithm replaces the old features with new ones. Features are represented as rectangles, where the rectangle for each feature contains the individuals having that feature set to **true**. Naturally, the individuals in the intersection of two rectangles have both features set to **true**. f_1 and f_2 have a big intersection, showing that they co-occur frequently. On the contrary, f_2 and f_5 have a small intersection, suggesting that their co-occurrence is less than that of the hazard (negatively correlated). f_3 is included in the intersection of f_1 and f_2, while f_4 has no common elements with any other. f_4 is incompatible with all of the others.

In the first iteration, f_1 and f_2 are combined and 3 features are created: $f_1 \wedge f_2$, $f_1 \wedge \overline{f_2}$ and $\overline{f_1} \wedge f_2$. These new features will replace f_1 and f_2, the original ones. At the second iteration, $f_1 \wedge f_2$ is combined with f_3. As f_3 is contained in $f_1 \wedge f_2$, the feature $\overline{f_1 \wedge f_2} \wedge f_3$ will have a support equal to zero and will be removed. Note that f_2 and f_5 are never combined, as they are considered uncorrelated. The final feature set will be $\{f_1 \wedge \overline{f_2}, f_1 \wedge f_2 \wedge f_3, f_1 \wedge f_2 \wedge \overline{f_3}, \overline{f_1} \wedge f_2, f_4, f_5\}$

3.1 **uFC** - the proposed algorithm

We define the set $P = \{p_1, p_2, ..., p_k\}$ of k user-supplied initial features and $I = \{i_1, i_2, ..., i_n\}$ the dataset described using P. We start from the hypothesis that even if the primitive set P cannot adequately describe the dataset I, there is a data-specific feature set $F = \{f_1, f_2, ..., f_m\}$ that can be created in order to represent the data better. New features are iteratively created using conjunctions of primitive features or their negations (as seen in Fig. 2). Our algorithm does not use the output of a learning algorithm in order to create the new features. Instead we use a greedy search strategy and a feature set evaluation function that can determine if a newly-obtained feature set is more appropriate than the former one.

The schema of our proposal is presented in Algorithm 2. The feature construction is performed starting from the dataset I and the primitives P. The algorithm follows the general inductive schema presented in Algorithm 1. At each iteration, **uFC** searches for frequently co-occurring pairs in the feature set created at the previous iteration (F_{iter-1}). It determines the candidate set O and then creates new features as conjunctions of the highest scoring pairs. The new features are added to the current set (F_{iter}), after which the set is filtered in

---

Algorithm 2 uFC - Unsupervised feature construction

Input: P – set of primitive user-given features
Input: I – the data expressed using P which will be used to construct features
  Inner parameters: λ – correlation threshold for searching, $limit\_iter$ – max no of iterations.
Output: F – set of newly-constructed features.
  $F_0 \leftarrow P$
  $iter \leftarrow 0$
  repeat
    $iter \leftarrow iter + 1$
    $O \leftarrow$ **search_correlated_pairs**$(I_{iter}, F_{iter-1}, \lambda)$
    $F_{iter} \leftarrow F_{iter-1}$
    while $O \neq \emptyset$ **do**
      $pair \leftarrow$ **highest_scoring_pair**(O)
      $F_{iter} \leftarrow F_{iter} \bigcup$ **construct_new_feat**$(pair)$
      remove_candidate$(O, pair)$
    end while
    prune_obsolete_features(F_{iter}, I_{iter})
    $I_{iter+1} \leftarrow$ **convert**(I_{iter}, F_{iter})
  until $F_{iter} = F_{iter-1}$ **OR** $iter = limit\_iter$
  $F \leftarrow F_{iter}$

---

order to remove obsolete features. At the end of each iteration, the dataset I is translated to reflect the feature set F_{iter}. A new iteration is performed as long as new features were generated in the current iteration and a maximum number of iterations have not yet been reached ($limit\_iter$ is a parameter for the algorithm).

3.2 Searching co-occurring pairs

The **search_correlated_pairs** function searches for frequently co-occurring pairs of features in a feature set F. We start with an empty set $O \leftarrow \emptyset$ and we investigate all possible pairs of features $\{f_i, f_j\} \in F \times F$. We use a function (r) to measure the co-occurrence of a pair of features $\{f_i, f_j\}$ and compare it to a threshold λ. If the value of the function is above the threshold, then their co-occurrence is considered as significant and the pair is added to O. Therefore, O will be

$$O = \{\{f_i, f_j\} | \forall \{f_i, f_j\} \in F \times F \, so \, that \, \mathbf{r}(\{f_i, f_j\}) > \lambda\}$$

The r function is the empirical **Pearson correlation coefficient**, which is a measure of the strength of the linear dependency between two variables. $r \in [-1, 1]$ and it is defined as the covariance of the two variables divided by the product of their standard deviations. The sign of the r function gives the direction of the correlation (inverse correlation for $r < 0$ and positive correlation for $r > 0$), while the absolute value or the square gives the strength of the correlation. A value of 0 implies that there is no linear correlation between the variables. When applied to Boolean variables, having the contingency table as shown in Table 1, the r function has the following formulation:

$$r(\{f_i, f_j\}) = \frac{a \times d - b \times c}{\sqrt{(a+b) \times (a+c) \times (b+d) \times (c+d)}}$$

The λ threshold parameter will serve to fine-tune the number of selected pairs. Its impact on the behaviour of the algorithm will be studied in Section 5.3. A method of automatic choice of λ using statistical hypothesis testing is presented in Section 6.1.

Table 1: Contingency table for two Boolean features

	f_j	$\neg f_j$
f_i	a	b
$\neg f_i$	c	d

3.3 Constructing and pruning features

Once O is constructed, **uFC** performs a greedy search. The function **highest_scoring_pair** is iteratively used to extract from O the pair $\{f_i, f_j\}$ that has the highest co-occurrence score.

The function **construct_new_feat** constructs three new features: $f_i \wedge f_j$, $\overline{f_i} \wedge f_j$ and $f_i \wedge \overline{f_j}$. They represent, respectively, the intersection of the initial two features and the relative complements of one feature in the other. The new features are guaranteed by construction to be negatively correlated. If one of them is set to true for an individual, the other two will surely be false. At each iteration, very simple features are constructed: conjunctions of two literals. The creation of more complex and semantically rich features appears through the iterative process. f_i and f_j can be either primitives or features constructed in previous iterations.

After the construction of features, the **remove_candidate** function removes from O the pair $\{f_i, f_j\}$, as well as any other pair that contains f_i of f_j. When there are no more pairs in O, **prune_obsolete_features** is used to remove from the feature set two types of features:

– **features that are false for all individuals**. These usually appear in the case of hierarchical relations. We consider that f_1 and f_2 have a hierarchical relation if all individuals that have feature f_1 true, automatically have feature f_2 true (e.g., f_1 "is a type of" f_2 or f_1 "is a part of" f_2). One of the generated features (in the example $\overline{f_1} \wedge f_2$) is false for all individuals and, therefore, eliminated. In the example of *water* and *cascade*, we create only *water* $\wedge$ *cascade* and *water* $\wedge \neg$*cascade*, since there cannot exist a cascade without water (considering that a value of **false** means the absence of a feature and not missing data).

– **features that participated in the creation of a new feature**. Effectively, all $\{f_i | \{f_i, f_j\} \in O, f_j \in F\}$ are replaced by the newly-constructed features.

$$\{f_i, f_j | f_i, f_j \in F, \{f_i, f_j\} \in O\} \xrightarrow{repl.} \{f_i \wedge f_j, \overline{f_i} \wedge f_j, f_i \wedge \overline{f_j}\}$$

4 Evaluation of a feature set

To our knowledge, there are no widely accepted measures to evaluate the overall correlation between the features of a feature set. We propose a measure inspired from the "inclusion-exclusion" principle (Feller, 1950). In set theory, this principle permits to express the cardinality of the finite reunion of finite ensembles by considering the cardinality of those ensembles and their intersections. In the Boolean form, it is used to estimate the probability of a *clause* (disjunction of literals) as a function of its composing *terms* (conjunctions of literals).

Given the feature set $F = \{f_1, f_2, ..., f_m\}$, we have:

$$p(f_1 \vee f_2 \vee ... \vee f_m) = \sum_{k=1}^{m} \left((-1)^{k-1} \sum_{1 \le i_1 < ... < i_k \le m} p(f_{i_1} \wedge f_{i_2} \wedge ... \wedge f_{i_k}) \right)$$

which, by putting apart the first term, is equivalent to:

$$p(f_1 \vee f_2 \vee ... \vee f_m) = \sum_{i=1}^{m} p(f_i) + \sum_{k=2}^{m} \left((-1)^{k-1} \sum_{1 \leq i_1 < ... < i_k \leq m} p(f_{i_1} \wedge f_{i_2} \wedge ... \wedge f_{i_k}) \right)$$

Without loss of generality, we can consider that each individual has at least one feature set to **true**. Otherwise, we can create an artificial feature "null" that is set to **true** when all the others are **false**. Consequently, the left side of the equation is equal to 1. On the right side, the second term is the probability of intersections of the features. Knowing that $1 \leq \sum_{i=1}^{m} p(f_i) \leq m$, this probability of intersection has a value of zero when all features are incompatible (no overlapping). It has a "worst case scenario" value of $m-1$, when all individuals have all the features set to **true**.

Based on these observations, we propose the **Overlapping Index** evaluation measure:

$$OI(F) = \frac{\sum_{i=1}^{m} p(f_i) - 1}{m - 1}$$

where $OI(F) \in [0,1]$ and "better" towards zero. Hence, a feature set F_1 describes a dataset better than another feature set F_2 when $OI(F_1) < OI(F_2)$.

4.1 Complexity of the feature set.

Number of features Considering the case of the majority of machine learning datasets, where the number of primitives is inferior to the number of individuals in the dataset, reducing correlations between features comes at the expense of increasing the number of features. Consider the pair of features $\{f_i, f_j\}$ judged correlated. Unless $f_i \supseteq f_j$ or $f_i \subseteq f_j$, the algorithm will replace $\{f_i, f_j\}$ by $\{f_i \wedge f_j, \overline{f_i} \wedge f_j, f_i \wedge \overline{f_j}\}$, thus increasing the total number of features. A feature set that contains too many features is no longer informative, nor comprehensible. The maximum number of features that can be constructed is mechanically limited by the number of unique combinations of primitives in the dataset (the number of unique individuals).

$$|F| \leq unique(I) \leq |I|$$

where F is the constructed feature set and I is the dataset.

To measure the complexity in terms of number of features, we use:

$$C_0(F) = \frac{|F| - |P|}{unique(I) - |P|}$$

where P is the primitive feature set. C_0 measures the ratio between how many extra features are constructed and the maximum number of features that can be constructed. $0 \leq C_0 \leq 1$ and a value closer to 0 means a feature set less complex.

The average length of features At each iteration, simple conjunctions of two literals are constructed. Complex Boolean formulas are created by combining features constructed in previous iterations. Long and complicated expressions generate incomprehensible features, which are more likely a random side-effect rather than a product of underlying semantics.

We define C_1 as the average number of literals (a primitive or its negation) that appear in a Boolean formula representing a new feature.

$$\overline{P} = \{\overline{p_i} | p_i \in P\}; \quad \mathscr{L} = P \bigcup \overline{P}$$

$$C_1(F) = \frac{\sum_{f_i \in F} |\{l_j | l_j \in \mathscr{L}, l_j \, appears \, in \, f_i\}|}{|F|}$$

where P is the primitive set and $1 \leq C_1 < \infty$.

As more iterations are performed, the feature set contains more features (C_0 grows) which are increasingly more complex (C_1 grows). This suggests a correlation between the two. What is more, since C_1 can potentially double at each iteration and C_0 can have at most a linear increase, the correlation is exponential. For this reason, in the following sections we shall use only C_0 as the complexity measure.

Overfitting All algorithms that learn from data risk overfitting the solution to the learning set. There are two ways in which **uFC** can overfit the resulted feature set, corresponding to the two complexity measures above: a) constructing too many features (measure C_0) and b) constructing features that are too long (measure C_1). The worst overfitting of type a) is when the algorithm constructs as many features as the maximum theoretical number (one for each individual in the dataset). The worst overfitting of type b) appears in the same conditions, where each constructed feature is a conjunction of all the primitives appearing for the corresponding individual. The two complexity measures can be used to quantify the two types of ovefitting. Since C_0 and C_1 are correlated, both types of overfitting appear simultaneously and can be considered as two sides of a single phenomenon.

4.2 The trade-off between two opposing criteria

C_0 is a measure of how overfitted a feature set is. In order to avoid overfitting, feature set complexity should be kept at low values, while the algorithm optimizes the co-occurrence score of the feature set. Optimizing both the correlation score and the complexity at the same time is not possible, as they are opposing criteria. A compromise between the two must be achieved. This is equivalent to the optimization of two contrary criteria, which is a very well-known problem in multi-objective optimization. To acquire a trade-off between the two mutually contradicting objectives, we use the concept of **Pareto optimality** (Sawaragi et al, 1985), originally developed in economics. Given multiple feature sets, a set is considered to be Pareto optimal if there is no other set that has both a better correlation score and a better complexity for a given dataset. Pareto optimal feature sets will form the Pareto front. This means that no single optimum can be constructed, but rather a class of optima, depending on the ratio between the two criteria.

We plot the solutions in the plane defined by the complexity, as one axis, and the co-occurrence score, as the other. Constructing the Pareto front in this plane makes a visual evaluation of several characteristics of the **uFC** algorithm possible, based on the deviation of solutions compared to the front. The distance between the different solutions and the constructed Pareto front visually shows how stable the algorithm is. The convergence of the algorithm can be visually evaluated by how fast (in number of performed iterations) the algorithm transits the plane from the region of solutions with low complexity and high co-occurrence score to solutions with high complexity and low co-occurrence. We can visually evaluate overfitting, which corresponds to the region of the plane with high complexity and low co-occurence score. Solutions found in this region are overfitted.

In order to avoid overfitting, we propose the **"closest-point"** heuristic for finding a compromise between OI and C_0. We consider the two criteria to have equal importance. We consider as a good compromise, the solution in which the gain in co-occurrence score and the loss in complexity are fairly equal. If one of the indicators has a value considerably larger than the other, the solution is considered to be unsuitable. Such solutions would have either
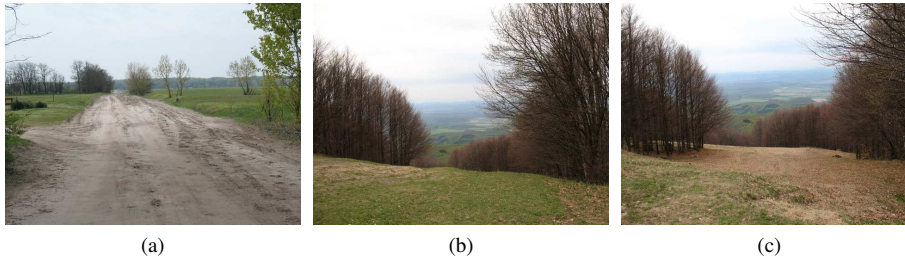
Fig. 3: Images related to the newly-constructed feature $sky \wedge \overline{building} \wedge panorama$ on `hungarian`: (a) Hungarian puszta, (b)(c) Hungarian Mátra mountains

a high correlation between features or a high complexity. Therefore, we perform a battery of tests and we search *a posteriori* the Pareto front for solutions for which the two indicators have essentially equal values. In the space of solutions, this translates into a minimal Euclidian distance between the solution and the ideal point (the point $(0;0)$).

5 Initial Experiments

Throughout the experiments, **uFC** was executed by varying only the two parameters: λ and $limit_{iter}$. We denote an execution with specific values for parameters as **uFC**(λ, $limit_{iter}$), whereas the execution where the parameters were determined *a posteriori* using the "closest-point" strategy will be noted **uFC***(λ, $limit_{iter}$). For **uFRINGE**, the maximum number of features was set at 300. We perform a comparative evaluation of the two algorithms seen from a qualitative and quantitative point of view, together with examples of typical executions. Finally, we study the impact of the two parameters of **uFC**.

Experiments were performed on three Boolean datasets. The `hungarian` dataset[1] is a real-life collection of images, depicting Hungarian urban and countryside settings. Images were manually tagged using one or more of the 13 tags. Each tag represents an object that appears in the image (eg. tree, cascade etc.). The tags serve as features and a feature takes the value **true** if the corresponding object is present in the image or **false** otherwise. The resulted dataset contains 264 individuals, described by 13 Boolean features. Once the dataset was constructed, the images were not used any more. The `street` dataset[2] was constructed in a similar way, starting from images taken from the LabelMe dataset (Russell et al, 2008). 608 urban images from Barcelona, Madrid and Boston were selected. Image labels were transformed into tags depicting objects by using the uniformization list provided with the toolbox. The dataset contains 608 individuals, described by 66 Boolean features.

The third dataset is "Spect Heart"[3] from the UCI. The corpus is provided with a "class" attribute and divided into a learning corpus and a testing one. We eliminated the class attribute and concatenated the learning and testing corpus into a single dataset. It contains 267 instances described by 22 Boolean features. Unlike the first two datasets, the features of `spect` have no specific meaning, being called "F1", "F2", ... , "F22".

---

[1] http://eric.univ-lyon2.fr/~arizoiu/files/hungarian.txt

[2] http://eric.univ-lyon2.fr/~arizoiu/files/street.txt

[3] http://archive.ics.uci.edu/ml/datasets/SPECT+Heart

<center>(a)   (b)   (c)</center>
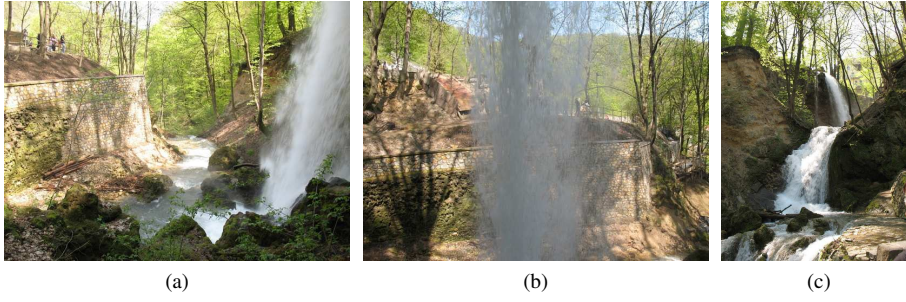
Fig. 4: Images related to the newly-constructed feature $water \wedge cascade \wedge tree \wedge forest$ on `hungarian`

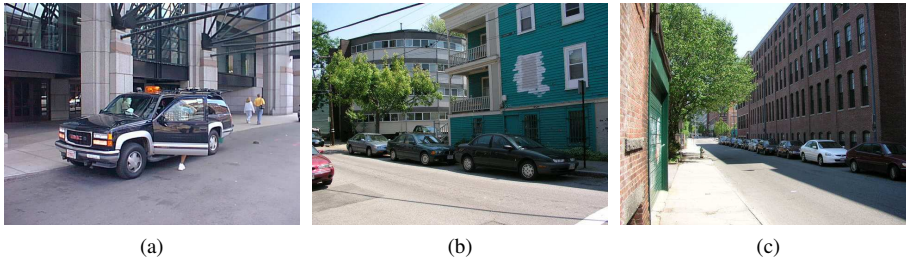<center>(a)   (b)   (c)</center>

Fig. 5: Images related to the newly-constructed feature $headlight \wedge windshield \wedge \overline{arm} \wedge head$ on `street`

5.1 uFC and uFRINGE: Qualitative evaluation

For the human reader, it is quite obvious why *water* and *cascade* have the tendency to appear together or why *road* and *interior* have the tendency to appear separately. One would expect, that based on a given dataset, the algorithms would succeed in making these associations and catching the underlying semantics. Table 2 shows the features constructed with **uFRINGE** and **uFC*** $(0.194, 2)$ on `hungarian`. A quick overview shows that constructed features manage to make associations that seem "logical" to a human reader. For example, one would expect the feature $sky \wedge \overline{building} \wedge panorama$ to denote images where there is a panoramic view and the sky, but no buildings, therefore suggesting images outside the city. Fig. 3 supports this expectation. Similarly, the feature $\overline{sky} \wedge building \wedge groups \wedge road$ covers urban images, where groups of people are present and $water \wedge cascade \wedge tree \wedge forest$ denotes a cascade in the forest (Fig. 4).

Comprehension quickly deteriorates when the constructed feature set is overfitted, when the constructed features are too complex. The execution of **uFC** $(0.184, 5)$ reveals features like:

$$\overline{\overline{sky} \wedge \overline{building}} \wedge tree \wedge building \wedge \overline{forest} \wedge \overline{\overline{sky} \wedge building \wedge groups \wedge road} \wedge$$
$$\overline{sky \wedge \overline{building} \wedge \overline{panorama}} \wedge \overline{\overline{groups} \wedge road} \wedge person \wedge \overline{\overline{sky}} \wedge groups \wedge road$$

Table 2: Feature sets constructed by **uFC** and **uFRINGE**

primitives	uFRINGE	uFC(0.194, 2)
person	$water \land forest \land grass \land water \land person$	$groups \land \overline{road} \land interior$
groups	$\overline{panorama} \land \overline{building} \land forest \land grass$	$groups \land \overline{road} \land \overline{interior}$
water	$tree \land person \land grass$	$groups \land \overline{road} \land interior$
cascade	$tree \land person \land \overline{grass}$	$water \land cascade \land tree \land forest$
sky	$\overline{groups} \land tree \land person$	$water \land cascade \land tree \land forest$
tree	$\overline{person} \land interior$	$water \land cascade \land tree \land forest$
grass	$\overline{person} \land interior$	$sky \land building \land \overline{tree \land forest}$
forest	$water \land \overline{panorama} \land grass \land \overline{groups} \land tree$	$sky \land building \land tree \land \overline{forest}$
statue	$water \land \overline{panorama} \land grass \land \overline{groups} \land tree$	$sky \land building \land tree \land forest$
building	$\overline{statue \land \overline{groups}} \land groups$	$sky \land \overline{building} \land panorama$
road	$\overline{statue \land \overline{groups}} \land \overline{groups}$	$sky \land \overline{building} \land \overline{panorama}$
interior	$\overline{panorama} \land \overline{statue} \land \overline{groups}$	$sky \land \overline{building} \land panorama$
panorama	$\overline{grass} \land \overline{water} \land \overline{forest} \land sky$	$\overline{groups} \land road \land person$
	$\overline{grass} \land \overline{water} \land \overline{forest} \land sky$	$\overline{groups} \land road \land \overline{person}$
	$person \land \overline{grass} \land \overline{water} \land \overline{forest}$	$\overline{groups} \land road \land person$
	$groups \land \overline{sky} \land \overline{grass} \land \overline{building}$	$sky \land building \land groups \land road$
	$groups \land \overline{sky} \land \overline{grass} \land \overline{building}$	$\overline{sky} \land building \land groups \land road$
	$groups \land \overline{person} \land \overline{water} \land \overline{forest} \land \overline{statue} \land groups$	$\overline{sky} \land building \land groups \land road$
	$groups \land \overline{person} \land \overline{water} \land \overline{forest} \land \overline{statue} \land groups$	$water \land cascade$
	$grass \land \overline{person} \land \overline{statue}$	$\overline{tree} \land forest$
	$grass \land \overline{person} \land \overline{statue}$	$grass$
	`... and 284 others`	$statue$

Even if the formula is not in the Disjunctive Normal Form (DNF), it is obvious that it is too complex to make any sense. If **uFC** tends to construct overly complex features, **uFRINGE** suffers from another type of dimensionality curse. Even if the complexity of features does not impede comprehension, the fact that there are over 300 hundred features constructed from 13 primitives makes the newly-constructed feature set unusable. The number of features is actually greater than the number of individuals in the dataset, which proves that some of the features are redundant. The actual correlation score of the newly-created feature set is even greater than the initial primitive set. What is more, new features present redundancy, just as predicted in section 2. For example, the feature *water* $\land$ *forest* $\land$ *grass* $\land$ *water* $\land$ *person* which contains two times the primitive *water*.

The same conclusions are drawn from execution on the `street` dataset. **uFC***$(0.322, 2)$ creates comprehensible features. For example *headlight* $\land$ *windshield* $\land \overline{arm} \land$ *head* (Fig. 5) suggests images in which the front part of cars appear. It is especially interesting how the algorithm specifies $\overline{arm}$ in conjunction with *head* in order to differenciate between people (*head* $\land$ *arm*) and objects that have heads (but no arms).

5.2 **uFC** and **uFRINGE**: Quantitative evaluation

Table 3 shows, for the three datasets, the values of certain indicators, like the size of the feature set, the average length of a feature (C_1), the *OI* and C_0 indicators. For each dataset, we compare four feature sets: the initial feature set (primitives), the execution of **uFC*** (parameters determined by the "closest-point" heuristic), **uFC** with another random set of

Table 3: Values of indicators for multiple runs on each dataset

	Strategy	#$feat$	$length$	OI	C_0
hungar.	**Primitives**	13	1.00	0.24	0.00
	uFC*(0.194, 2)	21	2.95	0.08	0.07
	uFC(0.184, 5)	36	11.19	0.03	0.20
	uFRINGE	306	3.10	0.24	2.53
street	**Primitives**	66	1.00	0.12	0.00
	uFC*(0.446, 3)	81	2.14	0.06	0.04
	uFC(0.180, 5)	205	18.05	0.02	0.35
	uFRINGE	233	2.08	0.20	0.42
spect	**Primitives**	22	1.00	0.28	0.00
	uFC*(0.432, 3)	36	2.83	0.09	0.07
	uFC(0.218, 4)	62	8.81	0.03	0.20
	uFRINGE	307	2.90	0.25	1.45

parameters and **uFRINGE**. For the `hungarian` and `street` datasets, the same parameter combinations are used as in the qualitative evaluation.

On all three datasets, **uFC*** creates feature sets that are less correlated than the primitive sets, while the increase in complexity is only marginal. Very few (2-3) iterations are needed, as **uFC** converges very fast. Increasing the number of iterations has very little impact on OI, but results in very complex vocabularies (large C_0 and feature lengths). In the feature set created by **uFC**$(0.180, 5)$ on `street`, on average, each feature contains more than 18 literals. This is obviously too much for human comprehension.

For **uFRINGE**, the OI indicator shows very marginal or no improvement on `spect` and `hungarian` datasets, and even a degradation on `street` (compared to the primitive set). Features constructed using this approach have an average length between 2.08 and 3.1 literals, just as much as the selected **uFC*** configuration. But, it constructs between 2.6 and 13.9 times more features than **uFC***. We consider this to be due to the lack of filtering in **uFRINGE**, which would also explain the low OI score. Old features remain in the feature set and amplify the total correlation by adding the correlation between old and new features.

5.3 Impact of parameters λ and $limit_{iter}$

In order to understand the impact of parameters, we executed **uFC** with a wide range of values for λ and $limit_{iter}$ and studied the evolution of the indicators OI and C_0. For each dataset, we varied λ between 0.002 and 0.5 with a step of 0.002. For each value of λ, we executed **uFC** by varying $limit_{iter}$ between 1 and 30 for the `hungarian` dataset, and between 1 and 20 for `street` and `spect`. We study the evolution of the indicators as a function of $limit_{iter}$, respectively λ, we plot the solution in the (OI, C_0) space and construct the Pareto front.

For the study of $limit_{iter}$, we hold λ fixed at various values and we vary only $limit_{iter}$. The evolution of the OI correlation indicator is given in Fig. 6a. As expected, the measure ameliorates with the number of iterations. OI has a very rapid descent and needs less than 10 iterations to converge on all datasets towards a value dependent on λ. The higher the value of λ, the higher the value of convergence. The complexity has a very similar evolution, but in the inverse direction: it increases with the number of iterations performed. It also converges
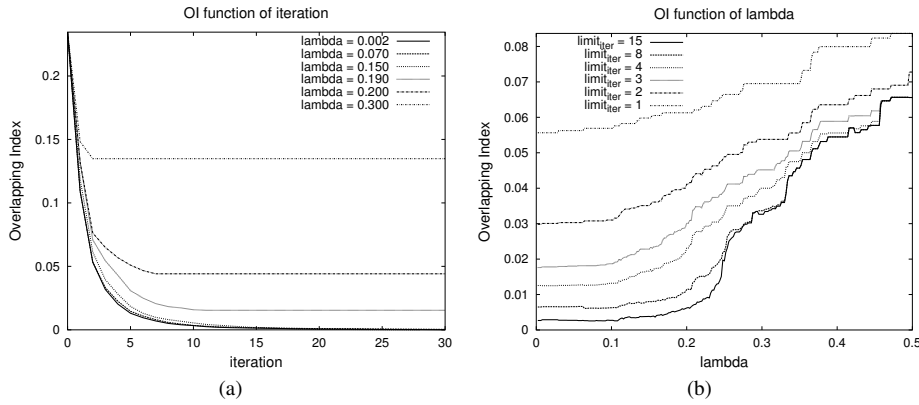
Fig. 6: Variation of *OI* indicator with $limit_{iter}$ on `hungarian` (a) and with λ on `street` (b)

towards a value that is dependent on λ: the higher the value of λ, the lower the complexity of the resulting feature set.

Similarly, we study λ by fixing $limit_{iter}$. Fig. 6b shows how *OI* evolves when varying λ. As foreseen, for all values of $limit_{iter}$, the *OI* indicator increases with λ, while C_0 decreases with λ. *OI* shows an abrupt increase between 0.2 and 0.3, for all datasets. For lower values of λ, many pairs get combined as their correlation score is bigger than the threshold. As λ increases, only highly correlated pairs get selected and this usually happens in the first iterations. Performing more iterations does not bring any change and indicators are less dependent on $limit_{iter}$. For `hungarian`, no pair has a correlation score higher than 0.4. Setting λ higher than this value causes **uFC** to output the primitive set (no features are created).
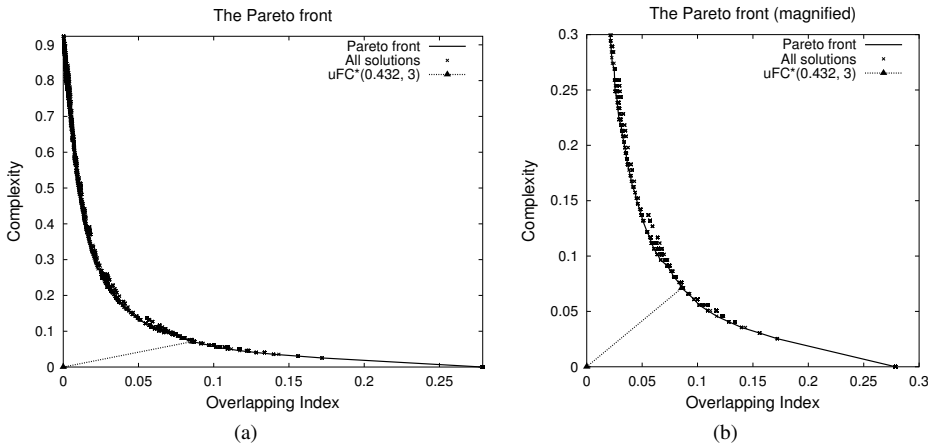

Fig. 7: The distribution of solutions, the Pareto front and the closest-point on `spect` dataset

To study Pareto optimality, we plot the generated solutions in the (OI, C_0) space. Fig. 7a presents the distribution of solutions, the Pareto front and the solution chosen by the "closest-point" heuristic. The solutions generated by **uFC** with a wide range of parameter values are not dispersed in the solution space, but their distribution is rather close together. This shows good algorithm stability. Even if not all the solutions are Pareto optimal, none of them are too distant from the front and there are no outliers.

Most of the solutions densely populate the part of the curve corresponding to low OI and high C_0. As pointed out in the Section 4.2, the area of the front corresponding to high feature set complexity (high C_0) represents the overfitting area. This confirms that the algorithm converges fast, then enters overfitting. Most of the improvement in quality is done in the first 2-3 iterations, while further iterating improves quality only marginally with the cost of an explosion of complexity. The "closest-point" heuristic keeps the constructing out of overfitting, by stopping the algorithm at the point where the gain of co-occurence score and the loss in complexity are fairly equal. Fig. 7b magnifies the region of the solution space corresponding for low numbers of iterations.

5.4 Relation between number of features and feature length

Both the average length of a feature (C_1) and the number of features (C_0) increase with the number of iterations. In Section 4.1 we have speculated that the two are correlated: $C_1 = f(C_0)$. For each λ in the batch of tests, we create the C_0 and C_1 series depending on the $limit_{iter}$ and we perform a statistical hypothesis test, using the Kendall rank coefficient as the test statistic. The Kendall rank coefficient is particularly useful as it makes no assumptions about the distributions of C_0 and C_1. For all values of λ, for all datasets, the statistical test revealed a p-value of the order of 10^{-9}. This is consistently lower than habitually used significance levels and makes us reject the null independence hypothesis and conclude that C_0 and C_1 are statistically dependent.

6 Improving the uFC algorithm

The major difficulty of **uFC**, shown by the initial experiments, is setting the values of parameters. An unfortunate choice would result in either an overly complex feature set or a feature set where features are still correlated. But both parameters λ and $limit_{iter}$ are dependent on the dataset and finding the suitable values would prove to be a process of trial and error for each new corpus. The "closest-point" heuristic achieves acceptable equilibrium between complexity and performance, but requires multiple executions with large choices of values for parameters and the construction of the Pareto front, which might not always be desirable or even possible.

We propose a new method for choosing λ based on statistical hypothesis testing and a new stopping criterion inspired from the "closest-point" heuristic. These will be integrated into a new "risk-based" heuristic that approximates the best solution while avoiding the time consuming construction of multiple solutions and the Pareto front. The only parameter is the significance level α, which is independent of the dataset, and makes the task of running **uFC** on new, unseen datasets easy. A pruning technique is also proposed.

6.1 Automatic choice of λ

We propose replacing the user-supplied co-occurrence threshold λ with a technique that selects only pairs of features for whom the positive linear correlation is statistically significant. These pairs are added to the set O of co-occurring pairs (defined in Section 3.2) and, starting from O, new features are constructed. We use a statistical method: the *hypothesis testing*. For each pair of candidate features, we test the independence hypothesis H_0 against the positive correlation hypothesis H_1.

We use as a test statistic the Pearson correlation coefficient (calculated as defined in Section 3.2) and test the following formally defined hypothesis: $H_0 : \rho = 0$ and $H_1 : \rho > 0$, where ρ is the theoretical correlation coefficient between two candidate features. We can show that in the case of Boolean variables, having the contingency table shown in Table 1, the observed value of the χ^2 of independence is $\chi^2_{obs} = nr^2$ (n is the size of the dataset). Consequently, considering true the hypothesis H_0, nr^2 is approximately following a χ^2 distribution with one degree of freedom ($nr^2 \sim \chi^2_1$), resulting in $r\sqrt{n}$ following a standard normal distribution ($r\sqrt{n} \sim N(0,1)$), given that n is large enough.

We reject the H_0 hypothesis in favour of H_1 if and only if $r\sqrt{n} \geq u_{1-\alpha}$, where $u_{1-\alpha}$ is the right critical value for the standard normal distribution. Two features will be considered significantly correlated when $r(\{f_i, f_j\}) \geq \frac{u_{1-\alpha}}{\sqrt{n}}$. The significance level α represents the risk of rejecting the independence hypothesis when it was in fact true. It can be interpreted as the *false discovery risk* in data mining. In the context of feature construction it is the *false construction risk*, since this is the risk of constructing new features based on a pair of features that are not really correlated. Statistical literature usually sets α at 0.05 or 0.01, but levels of 0.001 or even 0.0001 are often used.

The proposed method repeats the independence test a great number of times, which inflates the number of type I errors. Ge et al (2003) presents several methods for controlling the false discoveries. Setting aside the Bonferroni correction, often considered too simplistic and too drastic, one has the option of using sequential rejection methods (Benjamini and Liu, 1999; Holm, 1979), the q-value method of Storey (Storey, 2002) or making use of bootstrap (Lallich et al, 2006). In our case, applying these methods is not clear-cut, as tests performed at each iteration depend on the results of the tests performed at previous iterations. It is noteworthy that a trade-off must be acquired between the inflation of false discoveries and the inflation of missed discoveries. This makes us choose a risk between 5% and $\frac{5\%}{m}$, where m is the theoretical number of tests to be performed.

6.2 Candidate pruning technique. Stopping criterion.

Pruning In order to apply the χ^2 independence test, it is necessary that the expected frequencies considering true the H_0 hypothesis be greater or equal than 5. We add this constraint to the new feature search strategy (subsection 3.2). Pairs for whom the values of $\frac{(a+b)(a+c)}{n}$, $\frac{(a+b)(b+c)}{n}$, $\frac{(a+c)(c+d)}{n}$ and $\frac{(b+d)(c+d)}{n}$ are not greater than 5, will be filtered from the set of candidate pairs O. This will impede the algorithm from constructing features that are present for very few individuals in the dataset.

Risk-based heuristic We introduced in Section 4.2 the "closest-point" for choosing the values for parameters λ and $limit_{iter}$. It searches the solution on the Pareto front for which the indicators are sensibly equal. We transform the heuristic into a stopping criterion: *OI* and C_0 are combined into a single formula, the **root mean square** (RMS). The algorithm will
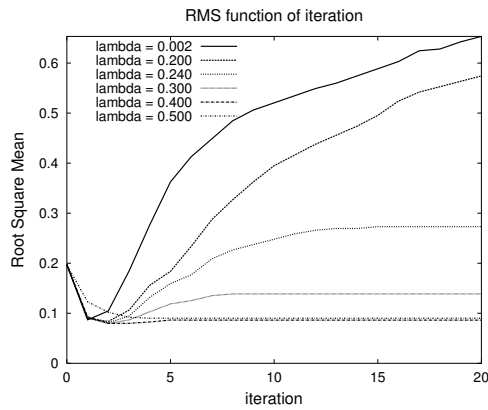
Fig. 8: RMS vs. $limit_{iter}$ on `spect`

stop iterating when RMS has reached a minimum. Using the generalized mean inequality, we can prove that $RMS(OI, C_0)$ has only one global minimum, as with each iteration the complexity increases and OI descends.

The $limit_{iter}$ parameter, which is data-dependent, is replaced by the automatic *RMS* stopping criterion. This stopping criterion together with the automatic λ choice strategy, presented in Section 6.1, form a data-independent heuristic for choosing parameters. We will call the new heuristic **risk-based heuristic**. This new heuristic will make it possible to approximate the best parameter compromise and avoid the time consuming task of computing a batch of solutions and constructing the Pareto front.

7 Further Experiments

We test the proposed ameliorations, similarly to what was shown in Section 5, on the same three datasets: `hungarian`, `spect` and `street`. We execute **uFC** in two ways: the classical **uFC** (Section 3) and the improved **uFC** (Section 6). The classical **uFC** needs to have parameters λ and $limit_{iter}$ set (noted **uFC**(λ, $limit_{iter}$)). **uFC***(λ, $limit_{iter}$) denotes the execution with parameters which were determined *a posteriori* using the "closest-point" heuristic. The improved **uFC** will be denoted as **uFC**$_\alpha$(*risk*). The "risk-based" heuristic will be used to determine the parameters and control the execution.

7.1 Risk-based heuristic for choosing parameters

Root Mean Square In the first batch of experiments, we study the variation of the Root Means Square aggregation function for a series of selected values of λ. We vary $limit_{iter}$ between 0 and 30, for `hungarian`, and between 0 and 20 for `spect` and `street`. The evolution of RMS is presented in Fig. 8.

For all λ the RMS starts by decreasing, as OI descends more rapidly than the C_0 increases. In just 1-3 iterations, RMS reaches its minimum and afterwards its value starts to increase. This is due to the fact that complexity increases rapidly, with only marginal improvement of quality. This behaviour is consistent with the results presented in Section 5. As
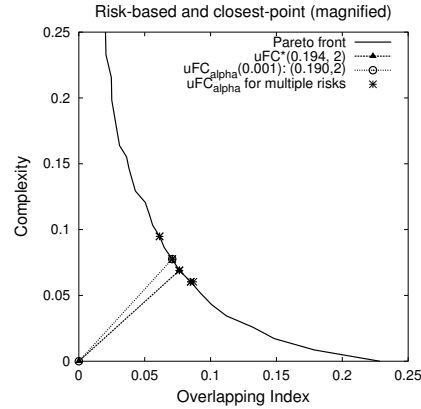
Fig. 9: "Closest-point" and "risk-based" for multiple α on `hungarian`

already discussed in Section 5.3, λ has a bounding effect over complexity, thus explaining why RMS reaches a maximum for higher values of λ.

The "risk-based" heuristic The second batch of experiments deals with comparing the "risk-based" heuristic to the "closest-point" heuristic. The "closest-point" was determined as described in Section 5. The "risk-based" heuristic was executed multiple times, with values for parameter $\alpha \in \{0.05, 0.01, 0.005, 0.001, 0.0008, 0.0005, 0.0003, 0.0001, 0.00005, 0.00001\}$

Table 4: "closest-point" and "risk-based" heuristics

	Strategy	λ	$limit_{iter}$	#feat	#common	length	OI	C_0
hung.	**Primitives**	-	-	13	-	1.00	0.235	0.000
	uFC*(0.194, 2)	0.194	2	21	19	2.95	0.076	0.069
	uFC$_\alpha$(0.001)	0.190	2	22		3.18	0.071	0.078
street	**Primitives**	-	-	66	-	1.00	0.121	0.000
	uFC*(0.446, 3)	0.446	3	87	33	2.14	0.062	0.038
	uFC$_\alpha$(0.0001)	0.150	1	90		1.84	0.060	0.060
spect	**Primitives**	-	-	22	-	1.00	0.279	0.000
	uFC*(0.432, 3)	0.432	3	36	19	2.83	0.086	0.071
	uFC$_\alpha$(0.0001)	0.228	2	39		2.97	0.078	0.086

Table 4 gives a quantitative comparison between the two heuristics. A risk of 0.001 is used for `hungarian` and 0.0001 for `spect` and `street`. The feature sets created by the two approaches are very similar, considering all indicators. Not only the differences between values for *OI*, C_0, average feature length and feature set dimension are negligible, but most of the created features are identical. On `hungarian`, 19 of the 21 features created by the two heuristics are identical. Table 5 shows the two features sets, with non-identical features in bold.

Fig. 9 presents the distribution of solutions created by the "risk-based" heuristic with multiple α, plotted on the same graphics as the Pareto front in the (*OI*, C_0) space. Solutions for different values of risk α are grouped closely together. Not all of them are on the

Table 5: Feature sets constructed by "closest-point" and "risk-based" heuristics on `hungarian`

primitives	uFC*(0.194, 2)	uFC$_\alpha$(0.001)
person	*groups* $\land \overline{road} \land$ *interior*	*groups* $\land \overline{road} \land$ *interior*
groups	*groups* $\land \overline{road} \land \overline{interior}$	*groups* $\land \overline{road} \land \overline{interior}$
water	*groups* $\land \overline{road} \land$ *interior*	*groups* $\land \overline{road} \land$ *interior*
cascade	*water* $\land$ *cascade* $\land$ *tree* $\land$ *forest*	*water* $\land$ *cascade* $\land$ *tree* $\land$ *forest*
sky	*water* $\land$ *cascade* $\land \overline{tree} \land$ *forest*	*water* $\land$ *cascade* $\land \overline{tree} \land$ *forest*
tree	$\overline{water} \land$ *cascade* $\land$ *tree* $\land$ *forest*	*water* $\land$ *cascade* $\land$ *tree* $\land$ *forest*
grass	*sky* $\land$ *building* $\land$ *tree* $\land \overline{forest}$	*sky* $\land$ *building* $\land$ *tree* $\land \overline{forest}$
forest	*sky* $\land$ *building* $\land$ *tree* $\land \overline{forest}$	*sky* $\land$ *building* $\land$ *tree* $\land \overline{forest}$
statue	$\overline{sky} \land$ *building* $\land$ *tree* $\land$ *forest*	$\overline{sky} \land$ *building* $\land$ *tree* $\land$ *forest*
building	*sky* $\land \overline{building} \land$ *panorama*	*sky* $\land \overline{building} \land$ *panorama*
road	*sky* $\land \overline{building} \land \overline{panorama}$	*sky* $\land \overline{building} \land \overline{panorama}$
interior	*sky* $\land \overline{building} \land$ *panorama*	*sky* $\land \overline{building} \land$ *panorama*
panorama	$\overline{groups} \land$ *road* $\land$ *person*	$\overline{groups} \land$ *road* $\land$ *person*
	$\overline{groups} \land$ *road* $\land \overline{person}$	$\overline{groups} \land$ *road* $\land \overline{person}$
	$\overline{groups} \land \overline{road} \land$ *person*	$\overline{groups} \land$ *road* $\land$ *person*
	water $\land \overline{cascade}$	$\overline{\textbf{sky}} \land \textbf{building} \land \textbf{groups} \land \textbf{road}$
	$\overline{\textbf{sky}} \land \textbf{building}$	$\overline{\textbf{sky}} \land \textbf{building} \land \overline{\textbf{groups}} \land \textbf{road}$
	$\overline{tree} \land$ *forest*	$\overline{\textbf{sky}} \land \textbf{building} \land \textbf{groups} \land \textbf{road}$
	$\textbf{groups} \land \textbf{road}$	*water* $\land \overline{cascade}$
	grass	$\overline{tree} \land$ *forest*
	statue	*grass*
		statue

Pareto front, but they are never too far from the "closest-point" solution, providing a good equilibrium between quality and complexity.
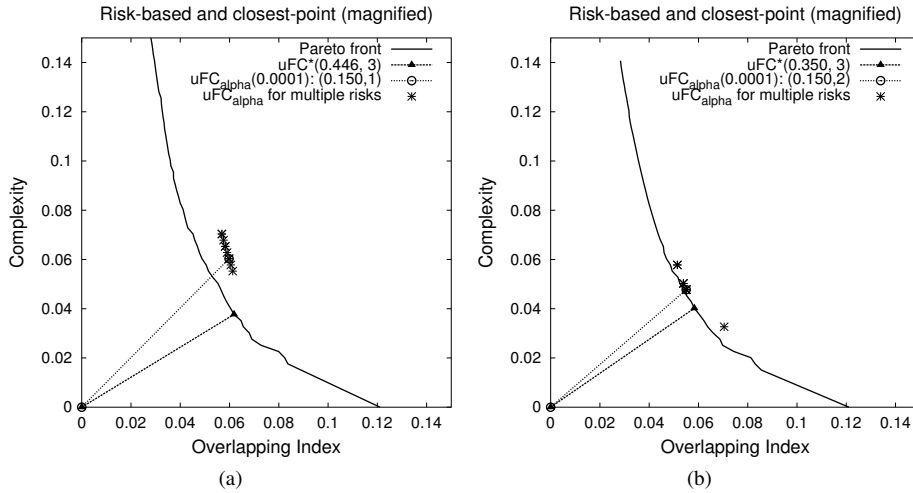

Fig. 10: "Closest-point" and "Risk-based" heuristics for `street` without pruning (a) and with pruning (b)

On street, performances of the "risk-based" heuristic start to degrade compared to
uFC*. Table 4 shows differences in the resulted complexity and only 33% of the constructed
features are common for the two approaches. Fig. 10a shows that solutions found by the
"risk-based" approach are moving away from the "closest-point". The cause is the large
size of the street dataset. As the sample size increases, the null hypothesis tends to be
rejected at lower levels of p-value. The auto-determined λ threshold is set too low and the
constructed feature sets are too complex. Pruning solves this problem as shown in Fig. 10b
and Section 7.2.

7.2 Pruning the candidates

The pruning technique is independent of the "risk-based" heuristic and can be applied in
conjunction with the classical **uFC** algorithm. An execution of this type will be denoted
$\mathbf{uFC}_P(\lambda, max_{iter})$. We execute $\mathbf{uFC}_P(\lambda, max_{iter})$ with the same parameters and on the same
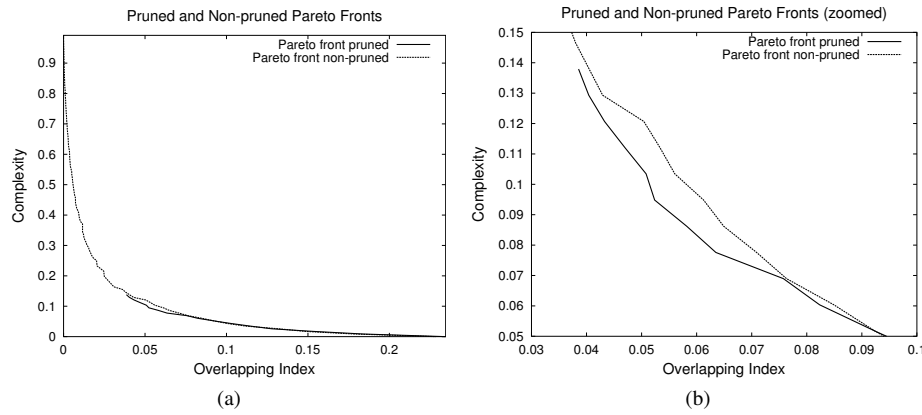datasets as described in Section 5.3.

Fig. 11: Pruned and Non-pruned Pareto Fronts on hungarian (a) and a zoom to the relevant
part (b)

We compare **uFC** with and without pruning by plotting on the same graphic the two
Pareto fronts resulted from each set of executions. Fig. 11a shows the pruned and non-
pruned Pareto fronts on hungarian. The graphic should be interpreted in a manner similar
to a ROC curve, since the algorithm seeks to minimize *OI* and C_0 at the same time. When one
Pareto front runs closer to the origin of the graphic $(0,0)$ than a second, it means that the
first dominates the second one and, thus, its corresponding approach yields better results.
For all datasets, the pruned Pareto front dominates the non-pruned one. The difference is
marginal, but proves that filtering improves results.

The most important conclusion is that filtering limits complexity. As the initial experi-
ments (Fig. 7a) showed, most of the non-pruned solutions correspond to very high complex-
ities. Visually, the Pareto front is tangent to the vertical axis (the complexity) and showing
complexities around $0.8 - 0.9$ (out of 1). On the other hand, the Pareto front corresponding

to the pruned approach stops, for all datasets, for complexities lower than 0.15. This proves that filtering successfully discards solutions that are too complex to be interpretable.

Last, but not least, filtering corrects the problem of automatically choosing λ for the "risk-based" heuristic on big datasets. We ran **uFC**$_P$ with risk $\alpha \in \{0.05, 0.01, 0.005, 0.001, 0.0008, 0.0005, 0.0003, 0.0001, 0.00005, 0.00001\}$. Fig. 10b presents the distributions of solutions found with the "risk-based pruned" heuristic on `street`. Unlike results without pruning (Fig. 10a), solutions generated with pruning are distributed closely to those generated by "closest-point" and to the Pareto front.

7.3 Algorithm stability

In order to evaluate the stability of the **uFC**$_\alpha$ algorithm, we introduce noise in the `hungarian` dataset. The percentage of noise varied between 0% (no noise) and 30%. Introducing a certain percentage $x\%$ of noise means that $x\% \times k \times n$ random features in the datasets are inverted (false becomes true and true becomes false). k is the number of primitives and n is the number of individuals. For each given noise percentage, 10 noised datasets are created and only the averages are presented. **uFC**$_\alpha$ is executed for all the noised datasets, with the same combination of parameters ($risk = 0.001$ and no filtering).

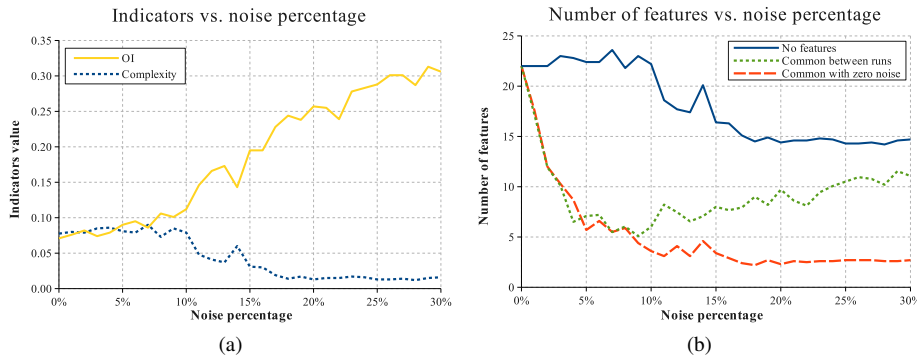Fig. 12: **uFC**$_\alpha$($risk$) stability on `hungarian` when varying the noise percentage: and indicators (a) and number of constructed features (b)

The stability is evaluated using five indicators:

– **Overlapping Index**;
– **Feature set complexity** (C_0);
– **Number of features**: the total number of features constructed by the algorithm;
– **Common with zero noise**: the number of identical features between the feature sets constructed based on the noised datasets and the non-noised dataset. This indicator evaluates the measure in which the algorithm is capable of constructing the same features, even in the presence of noise;
– **Common between runs**: the average number of identical features between feature sets constructed using datasets with the same noise percentage. This indicator evaluates how much the constructed feature sets differ at the same noise level.

As the noise percentage augments, the dataset becomes more random. Less pairs of primitives are considered as correlated and therefore less new features are created. Fig. 12a shows that the overlapping indicator increases with the noise percentage, while the complexity decreases. Furthermore, most features in the initial dataset are set to false. As the percentage of noise increases, the ratio equilibrates (more false values becoming true, than the contrary). As a consequence, for high noise percentages, the OI score is higher than for the primitive set.

The same conclusions can be drawn from Fig. 12b. The indicator **Number of features** descends when the noise percentage increases. This is because fewer features are constructed and the resulting feature set is very similar to the primitive set. The number of constructed features stabilizes around 20% of noise. This is the point where most of the initial correlation between features is lost. **Common with zero noise** has a similar evolution. The number of features identical to the non-noised dataset descends quickly and stabilizes around 20%. After 20%, all the identical features are among the initial primitives. Similarly, the value of **Common between runs** descends at first. For small values of introduced noise, the correlation between certain features is reduced, modifying the order in which pairs of correlated features are selected in Algorithm 2. This results in a diversity of constructed feature sets. As the noise level increases and the noised datasets become more random, the constructed feature sets resemble the primitive set, therefore augmenting the value of **Common between runs**.

8 Conclusion and future work

In this article, we propose two approaches towards feature construction. Unlike the other feature construction algorithms proposed so far in the literature, our proposals work in an unsupervised learning paradigm. **uFRINGE** is an unsupervised adaptation of the FRINGE algorithm, while **uFC** is a new approach that replaces linearly correlated features with conjunctions of literals. We prove that our approaches succeed in reducing the overall correlation in the feature set, while constructing comprehensible and interpretable features. We have performed extensive experiments to highlight the impact of parameters on the total correlation measure and feature set complexity. Based on the first set of experiments, we have proposed a heuristic that finds a suitable balance between quality and complexity and avoids time consuming multiple executions, followed by a Pareto front construction. We use statistical hypothesis testing and confidence levels for parameter approximation and reasoning on the Pareto front of the solutions for evaluation. We also propose a pruning technique, based on hypothesis testing, that limits the complexity of the generated features and speeds up the construction process.

For future development, we consider taking into account non-linear correlation between variables by modifying the metric of the search and the co-occurence measure. Another research direction will be adapting our algorithms for data of the Web 2.0 (e.g., automatic treatment of labels on the web). Several challenges arise, like very large label sets (it is common to have over 10 000 features), non-standard label names (see standardization preprocessing task that we have performed for the LabelMe dataset in Section 5) and missing data (a value of **false** can mean absence or missing data). We also consider converting generated features to the Disjunctive Normal Form for easier reading and suppressing features that have a low support in the dataset. This would reduce the size of the feature set by removing rare features, but would introduce new difficulties such as detecting nuggets.

References

Benjamini Y, Liu W (1999) A step-down multiple hypotheses testing procedure that controls the false discovery rate under independence. Journal of Statistical Planning and Inference 82(1-2):163–170

Blockeel H, De Raedt L, Ramon J (1998) Top-down induction of clustering trees. In: Proceedings of the 15th International Conference on Machine Learning, pp 55–63

Bloedorn E, Michalski RS (1998) Data-driven constructive induction. Intelligent Systems and their Applications 13(2):30–37

Cortes C, Vapnik V (1995) Support-vector networks. Machine learning 20(3):273–297

Dunteman GH (1989) Principal components analysis, vol 69. SAGE publications, Inc

Feller W (1950) An introduction to probability theory and its applications. Vol. I. Wiley

Ge Y, Dudoit S, Speed TP (2003) Resampling-based multiple testing for microarray data analysis. Test 12(1):1–77

Gomez G, Morales E (2002) Automatic feature construction and a simple rule induction algorithm for skin detection. In: Proc. of the ICML workshop on Machine Learning in Computer Vision, pp 31–38

Holm S (1979) A simple sequentially rejective multiple test procedure. Scandinavian journal of statistics pp 65–70

Huo X, Ni XS, Smith AK (2005) A survey of manifold-based learning methods. Mining of Enterprise Data pp 06–10

Lallich S, Rakotomalala R (2000) Fast feature selection using partial correlation for multi-valued attributes. In: Zighed DA, Komorowski J, Zytkow JM (eds) Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, LNAI Springer-Verlag, pp 221–231

Lallich S, Teytaud O, Prudhomme E (2006) Statistical inference and data mining: false discoveries control. In: COMPSTAT: proceedings in computational statistics: 17th symposium, Springer, p 325

Liu H, Motoda H (1998) Feature extraction, construction and selection: A data mining perspective. Springer

Matheus CJ (1990) Adding domain knowledge to sbl through feature construction. In: Proceedings of the Eighth National Conference on Artificial Intelligence, pp 803–808

Michalski RS (1983) A theory and methodology of inductive learning. Artificial Intelligence 20(2):111–161

Mo D, Huang SH (2011) Feature selection based on inference correlation. Intelligent Data Analysis 15(3):375–398

Motoda H, Liu H (2002) Feature selection, extraction and construction. Communication of IICM (Institute of Information and Computing Machinery) 5:67–72

Murphy PM, Pazzani MJ (1991) Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In: Proceedings of the Eighth International Workshop on Machine Learning, pp 183–187

Pagallo G, Haussler D (1990) Boolean feature discovery in empirical learning. Machine learning 5(1):71–99

Piatetsky-Shapiro G (1991) Discovery, analysis, and presentation of strong rules. Knowledge discovery in databases 229:229–248

Quinlan JR (1986) Induction of decision trees. Machine learning 1(1):81–106

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann

Russell BC, Torralba A, Murphy KP, Freeman WT (2008) Labelme: a database and web-based tool for image annotation. International Journal of Computer Vision 77(1):157–173

Sawaragi Y, Nakayama H, Tanino T (1985) Theory of multiobjective optimization, vol 176. Academic Press New York

Storey JD (2002) A direct approach to false discovery rates. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 64(3):479–498

Yang DS, Rendell L, Blix G (1991) A scheme for feature construction and a comparison of empirical methods. In: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, pp 699–704

Zheng Z (1995) Constructing nominal x-of-n attributes. In: Proceedings of International Joint Conference On Artificial Intelligence, vol 14, pp 1064–1070

Zheng Z (1996) A comparison of constructive induction with different types of new attribute. Tech. rep., School of Computing and Mathematics, Deakin University, Geelong

Zheng Z (1998) Constructing conjunctions using systematic search on decision trees. Knowledge-Based Systems 10(7):421–430