Akademie věd České republiky
Ústav teorie informace a automatizace

Academy of Sciences of the Czech Republic
Institute of Information Theory and Automation

RESEARCH REPORT

Somol P., Vácha P., Mikeš S., Hora J., Pudil P.*, Žid P.
Institute of Information Theory and Automation
Academy of Sciences of the Czech Republic

**Faculty of Management*
Prague University of Economics

Introduction to Feature Selection Toolbox 3 – The C++ Library for Subset Search, Data Modeling and Classification

No. 2287 October 2010

ÚTIA AV ČR, v.v.i., P.O. Box 18, 182 08 Prague, Czech Republic
E-mail: somol@utia.cas.cz
Fax: (+420)284683031

Introduction to Feature Selection Toolbox 3 – The C++ Library for Subset Search, Data Modeling and Classification

Petr Somol*, Pavel Vácha*, Stanislav Mikeš*, Jan Hora*, Pavel Pudil† and Pavel Žid*
*Dept. of Pattern Recognition, Institute of Information Theory and Automation
Czech Academy of Sciences, Pod vodárenskou věží 4, Prague 8, 18208
Email: see http://fst.utia.cz/?contacts
†Faculty of Management, Prague University of Economics
Email: pudil@fm.vse.cz

***Abstract*—We introduce a new standalone widely applicable software library for feature selection (also known as attribute or variable selection), capable of reducing problem dimensionality to maximize the accuracy of data models, performance of automatic decision rules as well as to reduce data acquisition cost. The library can be exploited by users in research as well as in industry. Less experienced users can experiment with different provided methods and their application to real-life problems, experts can implement their own criteria or search schemes taking advantage of the toolbox framework. In this paper we first provide a concise survey of a variety of existing feature selection approaches. Then we focus on a selected group of methods of good general performance as well as on tools surpassing the limits of existing libraries. We build a feature selection framework around them and design an object-based generic software library. We describe the key design points and properties of the library. The library is published at http://fst.utia.cz.**

***Keywords*-subset search; feature selection; attribute selection; variable selection; optimization; software library; machine learning; classification; pattern recognition;**

I. Introduction

Feature Selection (FS), also known as attribute selection or variable selection, can be considered one of the key stages in building automatic decision rules in machine learning, data modeling as well as in data mining [1]–[4]. FS reduces the dimensionality of input data by identifying such subset of original features, that captures the most (ideally all) information stored in the original data while consisting of possibly least redundant features.

By reducing noise and irrelevant parts of data, FS not only saves data acquisition cost and reduces data processing time, it also often improves accuracy of the constructed model or the recognition accuracy of the devised decision rule. This is possible due to the learning process being less affected by noise as well as due to reduced effects of the *curse of dimensionality* (term coined by R. Bellman), especially if the number of samples/records in training data is low with respect to data dimensionality. (With increasing dimensionality and constant number of data points the data space gets sparse and the possible clusters or boundaries between classes get less well defined.)

FS is common in computer supported decision making and data mining in various fields; in medical diagnostics to identify important symptoms, in economics and finance to identify credibility factors or to evaluate trends, in industry for defect detection, in document processing to identify terms that distinguish document categories, in security to identify important face image features, etc.

In this paper we describe our general purpose C++ feature selection library, Feature Selection Toolbox 3 (FST3), published at http://fst.utia.cz. In Sections II to III we give an overview of the existing feature selection methodology. In Section IV we introduce our implementation and explain its architecture. In Section V we discuss library usage and give code example. In concluding Sections VI to VII we discuss alternative solutions and future development. Note: diagrams in Figures 1, 3, 5 and 6 follow UML syntax. C++ class names in `italic` denote abstract classes.

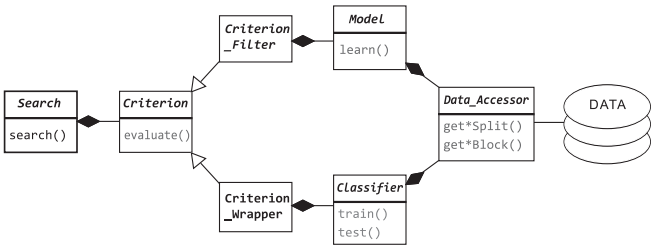

Figure 1. FST3 library architecture – simplified global overview. The call to `search()` returns a feature subset with respective criterion value

II. Feature Selection – Problem Formulation

Given a set Y of features (attributes, variables) of size $D = |\mathrm{Y}|$, let us denote $\mathcal{X}_d$ the set of all possible subsets of size d, where d represents the desired number of features. Let $J(\mathrm{X})$ be a criterion function that evaluates feature subset $\mathrm{X} \in \mathcal{X}_d$. Without any loss of generality, let us consider a higher value of J to indicate a better feature subset. Then the feature selection problem can be formulated as follows:

Find the subset $\tilde{X}_d$ for which

$$J(\tilde{\mathrm{X}}_d) = \max_{\mathrm{X} \in \mathcal{X}_d} J(\mathrm{X}). \quad (1)$$

Assuming that a suitable criterion function has been chosen to evaluate the effectiveness of feature subsets, feature selection is reduced to a search problem that detects an optimal feature subset based on the selected measure. Note that this is a potentionally expensive combinatorial problem as the number of candidate subsets is $\binom{D}{d}$.

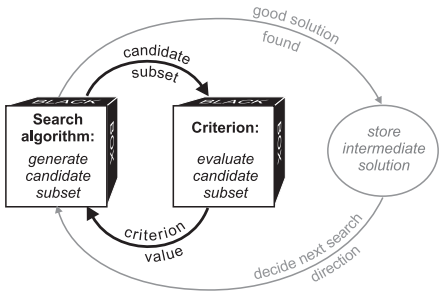

Figure 2. Feature subset selection methods can be generally viewed as black box procedures generating a sequence of candidate subsets with respective criterion values, among which intermediate solutions are chosen

Note that the choice of d may be a complex issue depending on problem characteristics, unless d can be optimized as part of the search process (see Sect. III-C).

III. Feature Selection – Review

The first step in solving the FS problem involves choosing appropriate FS tools based on the knowledge (or lack of therein) of available training data properties. The key decision to be made involves the choice of criterion (see Sect. III-A) and search algorithm (see Sect. III-B). The correct choice of FS tools for the given task is a complex issue [1], [4]–[7] while no single approach can be recommended as generally best. The search process itself can be viewed as an interative – potentially time consuming – operation (see Fig. 2). The search is stopped according to chosen stopping criterion; it can be defined in terms of achieved completness of search, criterion convergence threshold, subset size limit, time, etc. The common final step after the resulting feature subset has been obtained is validation on independent data (data not used in the course of search). In the following we give an overview over the existing variety of FS tools. The library design to be presented should reflect this variety and enable inclusion of any FS tool upon need (see Sect. IV).

A. Feature Subset Evaluation Criteria

In the course of a FS process individual features and/or feature subsets need to be evaluated using a *criterion* function. An optimal subset is always relative to a particular criterion. The ultimate FS criterion – *probability of error* – is difficult to evaluate directly [1]. It is therefore usually substituted by one of available alternatives either from the family of dependent or independent FS criteria.

Dependent FS criteria are based on estimating the error rate of a chosen machine learning tool, usually a classifier or regressor. Dependent criteria will thus favour subsets tightly coupled with a particular tool. The solutions obtained using dependent criteria may not be good when used in a different context, although for the given tool they are often favorable.

Independent FS criteria attempt to assess the merit of features from the data [1], [8]. Independent criteria can be roughly divided into the following categories [4]:

Distance measures are also known as separability, divergence or discrimination measures, because their purpose is to maximize the distance (and thus separability) between classes. Interclass distance can be simply evaluated using linear or nonlinear metrics (e.g., Minkowski, Euclidean, Chebychev, etc.). An assumption of data normality is often accepted to enable use of probabilistic distance measures (e.g., Mahalanobis, Bhattacharyya, Divergence, Patrick-Fischer), which are otherwise difficult to enumerate [1]. Many probabilistic distance measures (e.g., Mahalanobis, Bhattacharyya, Divergence, Patrick-Fischer) can be analytically simplified in the two-class case when the classes have parametric distributions [1].

Margin-based measures maximize the margin that separates classes [9], [10] or the distance between a hypothesis and the closest hypothesis that assigns alternative label to the given instance [11]–[15].

Information measures determine the information gain from a feature, or a mutual information between the feature and a class label [16], or class separability based on entropy [1]. The maximal-relevance-minimal-redundancy (mRMR) criterion has been shown to be equivalent to maximal statistical dependency of the target class on the data distribution, but more efficient [17]. Entropy also belongs to this category of indicators of class separability [1].

Dependence measures, also known as correlation measures, or similarity measures, quantify the ability to predict the value of one variable from the value of another [18]. The correlation coefficient can be used to find the correlation between a feature and a class [19], [20].

Consistency measures minimize the number of features that separate classes as consistently as the full feature set [21], [22]. An inconsistency is defined as two instances having the same feature values but different class labels.

The relation of various measures to the probability of error in terms of bounds has been widely studied [1], [23]–[28], but remains an important open problem.

It is common to distinguish FS methods according to the type of employed criteria [5]. *Wrappers* optimize the *dependent criteria* [5], [29]. *Filters* optimize the *independent criteria* [1]. Filters tend to generalize better and search faster than wrappers, but wrappers produce more accurate results for particular machine learning tools. *Hybrid methods*

attempt to combine the best of other approaches [4], [30], [31]. *Embedded methods* incorporate FS into modelling [32]–[34] and can be viewed as more effective but less general form of wrappers.

B. Feature Subset Search Algorithms

Provided a suitable FS criterion is chosen, the FS problem can be viewed as a *combinatorial subset search problem* (a form of pure binary programming problem in optimization terms). Due to the number of possible feature combinations the time complexity becomes an issue with increasing problem dimensionality.

With the exception of the exhaustive search, all ***optimal methods*** are based on Branch & Bound [35], [36] and can be used only with monotonic criteria [1] (the case of, e.g., many *distance measures*). All optimal methods are applicable to lower-dimensional problems only (roughly $D < 50$) due to their exponential complexity. FS in higher-dimensional problems and in problems with non-monotonous criteria is enabled by the polynomially complex ***sub-optimal methods***. Remark: Optimality with respect to chosen FS criterion does not necessarily imply better performance of the final system. Due to various negative problem-dependent effects (curse-of-dimensionality, insufficient sample-size-to-dimensionality ratio, sampling errors, estimation errors, etc.) it may well happen that sub-optimal solutions over-perform the optimal ones in terms of generalization [6]. Sub-optimally selected subsets may perform better if the learning process is unstable [7] or tends to over-fit [6] (see Sect. III-E).

Deterministic heuristic *sub-optimal methods* implement various forms of hill climbing to produce satisfactory results in polynomial time. Unlike *sequential selection* [1], *floating search* does not suffer from the nesting problem [37] and finds good solutions for each subset size [37], [38]. *Oscillating search* and *dynamic oscillating search* can improve existing solutions [39], [40]. Stochastic (randomized) methods like *random subspace* [41], *evolutionary algorithms* [42], *memetic algorithms* [43] or swarm algorithms like *ant colony* [44] may be better suited to over-come local extrema, yet may take longer to converge. The *Relief* algorithm [10] iteratively estimates feature weights according to their ability to discriminate between neighboring patterns. Deterministic search can be notably improved by randomization as in *simulated annealing* [45], *tabu search* [46], randomized *oscillating search* [39] or in combined methods [47].

The Best Individual Feature (BIF), or *individual feature ranking* approach is the fastest and simplest approach to FS, often the only applicable in problems of very high dimensionality. BIF is standard in text categorization [48], [49], genetics [50], [51], etc. BIF may be preferable not only because of speed but also to overcome FS stability and over-fitting problems (see Sect. III-E).

C. The Question of Feature Subset Size

Note that many standard FS criteria are monotonous with respect to subset size (e.g., probabilistic *distance measures* [1]) and thus make direct optimization of d impossible as the full feature set would always be yielded. Accordingly, most of the traditional FS methods are defined as d-parametrized, i.e., they require the user to decide what cardinality should the resulting feature subset have. In contrary, d-optimizing FS methods optimize both the feature subset size and subset contents at once, provided a non-monotonous criterion is used. *Dependent criteria* are particularly suitable for this purpose (see Sect. IV-C2 and IV-D5).

D. Feature Acquisition Cost

Standard FS criteria assess features only based on their impact on classification performance. In many tasks, however, the cost of measurement acquisition should be reflected (cheap measurements may be preferable to very expensive ones that improve overall system accuracy only negligibly). This can be done by including a penalization factor in the FS criterion [52]. Measurement cost can be reflected by means of a complementary criterion that is applied in the course of search only when the primary criterion permits choice among (almost) equal candidate subsets [53]. Considering cost of feature groups instead of single features may improve FS results [54].

E. Over-fitting and instability issues

In analogy to classifier over-training the FS process may over-select [6]. Over-selected features may lead to poor generalization, i.e., degraded performance of the devised machine learning system on previously unknown data. Unstable FS process may lead to similar problems [7], [55], [56]. Preventing these issues requires careful handling of the trade-off between simpler methods that over-fit less and more thorough optimizers capable of revealing more useful information about feature dependences. Especially with low-sample-size or high-dimensional data sets it may be advisable to resort to trivial search tools like BIF. It is also advisable to consider non-trivial estimators (cross-validation, leave-one-out, etc.) when estimating classification accuracy (see Sect. IV-B). However, unless stability or over-fitting problems occur, the more advanced search methods that better reflect relations among features are likely to produce better results.

IV. Implementing Feature Selection Library

FS method implementations can be found in many software libraries. Often they play only support role, or are intended for use within specific limited domain, or the implementation runs only in specific environment like Matlab (see Sect. VI). FST3 differs by its primary focus on the feature selection problem. As such it implements not only

techniques available elsewhere, but also many specialized non-standard tools, including:

- (Parallelized) highly effective subset search methods to tackle computational complexity,
- Wrappers, filters and hybrid methods, deterministic and/or randomized,
- Anti-overfitting measures: criteria ensembles, result regularization, stability and similarity evaluation, etc.,
- Templated highly effective and customizable C++ code.

In view of the vastness and diversity of known approaches to FS (cf. Sect. III) the FS library necessarily must be restricted in number of the implemented tools. Although the battery if initially implemented tools is to be sufficient enough to cover a possibly extensive variety of important FS scenarios, it is crucial for the library design to be general enough to enable extension in any of its aspects. In the following we discuss the architecture, currently implemented set of tools as well as usage scenarios of FST3 (see also http://fst.utia.cz/?fst3_arch).

A. FST3 Library Architecture

For simplified global overview of FST3 library architecture see Fig. 1. Key entities involved in FS process have their counterparts in respective FST3 classes *Search*, Subset, *Criterion*, *Classifier* and *Model*. All data access is abstracted through a specialization of *Data_Accessor* (see Sect. IV-B and Fig. 3 for details). Concrete search algorithms (specializations of *Search*, see Sect. III-B and Fig. 6 for details) yield a Subset object, maximizing a chosen criterion (specialization of *Criterion*, see Sect. IV-C and Fig. 5 for details). Note that gray boxes in Figs. 3, 5 and 6 suggest points of straightforward library extension. For details and examples of library usage see Sect. V.

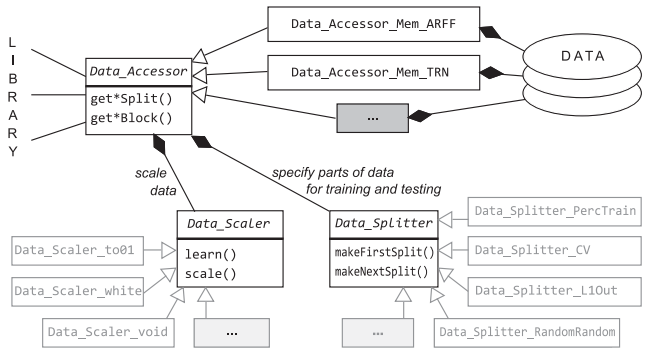

Figure 3. FST3 library architecture – simplified view of data access model

B. FST3 Data Access Layer

Access to data is realized through a *Data_Accessor* object. *Data_Accessor* provides standardized access to data for all library tools that need it, including *Criterion* and *Classifier* objects. Two data accessor specializations are provided, both caching the complete data in memory; Data_Accessor_Mem_ARFF reads data from ARFF files (standard Weka [57] machine library format), Data_Accessor_Mem_TRN reads data from a file in TRN format (see Fig. 8). Accessing different data sources (e.g., a database server) requires deriving a new class from *Data_Accessor* (see Fig. 3).

Two data access modifying mechanisms are implemented through dedicated objects: data pre-processing and multi-level data splitting to training/testing data parts (see Fig. 3).

Data_Scaler objects enable data preprocessing, primarily aimed at normalization of data values. Concrete *Data_Scaler* is called during *Data_Accessor* initialization. Default normalizers include Data_Scaler_to01 to scale all values to $[0, 1]$, Data_Scaler_white to normalize mean values and variances, and Data_Scaler_void to by-pass the mechanism.

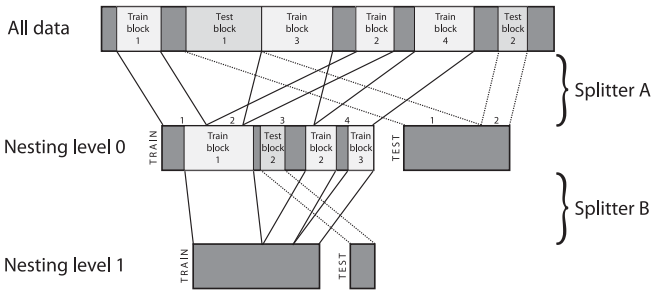

Figure 4. FST3 data access layer – nested train/test splitting

Data_Splitter objects enable sophisticated access to various parts of data at various stages of the FS process. The mechanism enables to realize various estimation schemes, selecting data parts for training/validation/testing, as well as FS stability evaluation [7], [55]. Splitters may provide multiple loops of data access (in k-fold cross-validation there would be k loops). The splitting mechanism allows nesting to arbitrary depth (see Fig. 4). Typically two nesting levels suffice; the "outer" spliting level is good for separating test data for final verification of FS results, the "inner" splitting level is good for estimating classifier accuracy in each step of wrapper-based FS selection process. Different *Data_Splitter* objects can be used in different nesting levels. Just one nesting level is set for data access at a time to hide the nesting mechanism from higher-level library tools (where simple access to training and testing data is expected) Each *Data_Splitter* object generates two lists of Data_Intervals (intervals indexing data samples), one to select data samples for training, second to select data samples for testing. Technically there is no restriction on possible interval overlapping (i.e., data sample re-use), nor on completeness of data coverage. See Fig. 4 for illustration. Default FST3 splitter objects implement cross-validation (class Data_Splitter_CV), hold-out (class Data_Splitter_Hold-

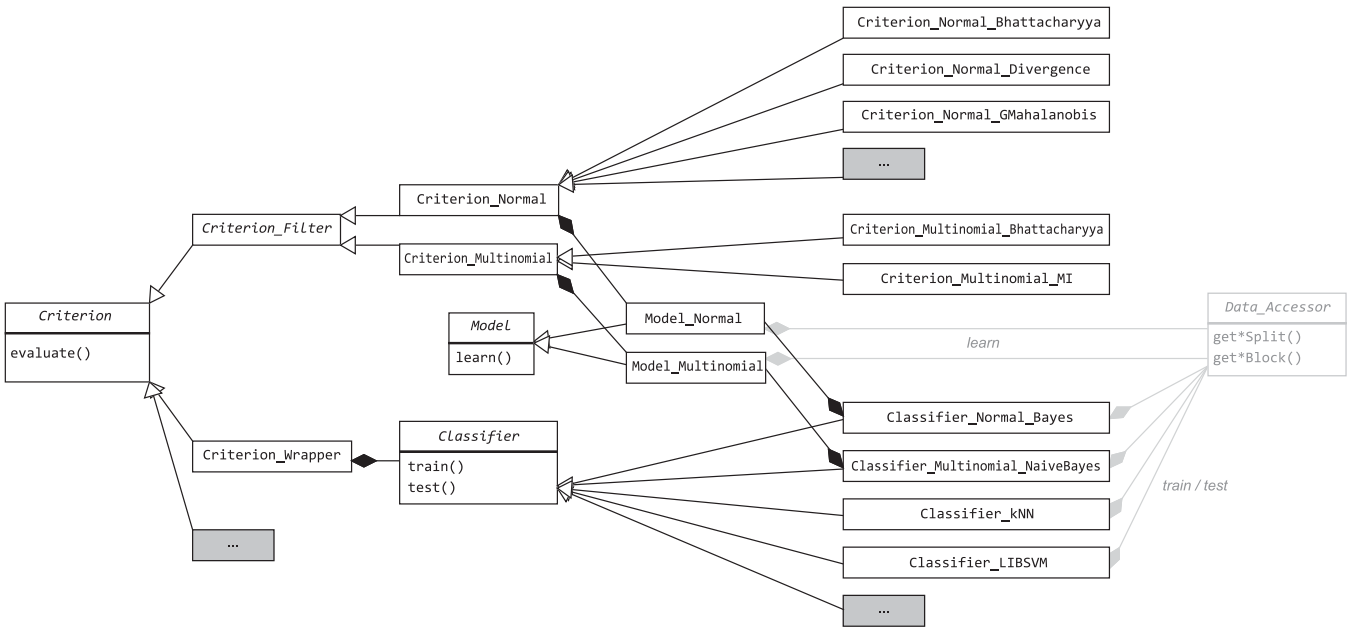

Figure 5. FST3 library architecture – subset evaluation criteria, data modeling and classifier classes

`Out`), leave-one-out (class `Data_Splitter_L1Out`), resubstitution (class `Data_Splitter_Resub`), random sampling (class `Data_Splitter_RandomRandom`).

Remark: splitting can be defined separately for each class or globally. Separate splitting leads to stratified data sampling (proportional with respect to class sizes). Global splitting enables, e.g., leave-one-out access where in one loop only one sample in one class is accessed for testing while all others in all other classes are accessed for training.

C. FST3 Feature Subset Evaluation Framework

See Figure 5 for overview of FST3 subset evaluation class hierarchy. Abstract class *`Criterion_Filter`* covers the implementations of *independent* criteria, class `Criterion_Wrapper` adapts *`Classifier`* objects to serve as *dependent* criteria. Several independent criteria and classifiers require first to estimate *`Model`* parameters from data.

1) Data Models: Class `Model_Normal` implements the multivariate gaussian (normal) model. Note that criteria based on normal model may yield misleading results for non-normally distributed data, especially in multi-modal case. Class `Model_Multinomial` implements the multinomial model, suitable esp. for text categorization [49]. `Model_Normal` is used in *`Criterion_Normal`*-based criteria and `Classifier_Normal_Bayes`. `Model_Multinomial` is used in *`Criterion_Multinomial`*-based criteria and `Criterion_Multinomial_NaiveBayes`.

2) Classifiers and Dependent Criteria: Class `Classifier_Normal_Bayes` implements Bayes classifier assuming normally distributed data, class `Classifier_kNN` implements k-Nearest Neighbor classifier, class `Classifier_LIBSVM` provides interface to externally linked Support Vector Machine library [58], class `Classifier_Multinomial_NaiveBayes` implements Naïve-like Bayes classifier assuming multinomially distributed data. Class `Criterion_Wrapper` adapts any object of type *`Classifier`* to serve as FS criterion, see Fig. 5.

3) Independent Criteria: Class `Criterion_Normal_Bhattacharyya` implements Bhattacharyya distance, `Criterion_Normal_GMahalanobis` implements generalized Mahalanobis distance, `Criterion_Normal_Divergence` implements the Divergence, all assuming normality of data. `Criterion_Multinomial_Bhattacharyya` implements multinomial Bhattacharyy distance, `Criterion_Multinomial_MI` implements individual Mutual Information.

D. FST3 Feature Subset Search Framework

See Figure 6 for overview of FST3 search algorithms' class hierarchy. Class `Search_BIF` implements *feature ranking* (cf. III-B) – the fastest and most trivial FS method that ignores possible inter-feature dependencies but runs in linear time. The more powerful sub-optimal *sequential search* methods generally follow the *hill-climbing* idea. To avoid local extremes the more advanced methods extend the hill-climbing idea by introducing various forms of backtracking and/or randomization. Sequential search methods do not guarantee optimality with respect to chosen FS criterion, yet they offer very good optimization-performance vs. (polynomial) time-complexity ratio, making them the favourable choice in FST3 (see Sect. IV-D1 to IV-D5). Optimal Branch & Bound-type methods (exponential time complexity) are to be included in next FST3 update.

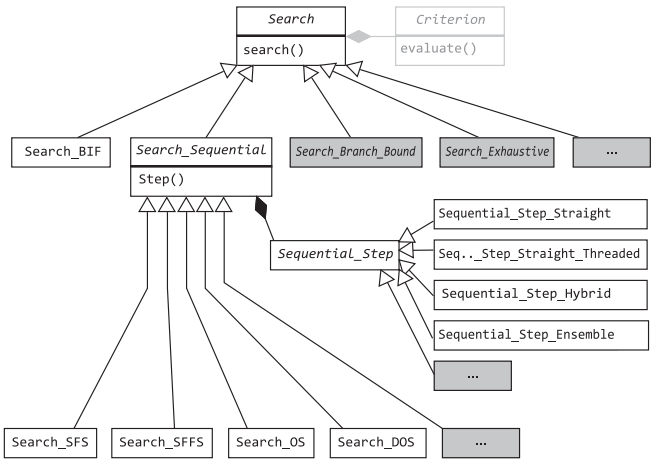

Figure 6. FST3 library architecture – search algorithm classes. Note that any sequential search algorithm can be parallelized, hybridized or turned to an ensemble selector by `Sequential_Step` specialization.

1) Sequential Search: Most of the known sequential FS algorithms (to be defined in the following) share the same "core mechanism" of adding and removing features to/from a current subset, that can be described as follows:

Definition 1: For a given current feature set X_d, let f^+ be the feature such that

$$f^+ = \arg\max_{f \in \mathrm{Y} \setminus \mathrm{X}_d} J^+(\mathrm{X}_d, f) \ ,$$

where $J^+(\mathrm{X}_d, f)$ denotes the criterion function used to evaluate the subset obtained by adding f ($f \in \mathrm{Y} \setminus \mathrm{X}_d$) to X_d. Then we shall say that $ADD(\mathrm{X}_d)$ is an operation of adding feature f^+ to the current set X_d to obtain set X_{d+1} if

$$ADD(\mathrm{X}_d) \equiv \mathrm{X}_d \cup \{f^+\} = \mathrm{X}_{d+1}, \qquad \mathrm{X}_d, \mathrm{X}_{d+1} \subset \mathrm{Y}.$$

Definition 2: For a given current feature set X_d, let f^- be the feature such that

$$f^- = \arg\max_{f \in \mathrm{X}_d} J^-(\mathrm{X}_d, f) \ ,$$

where $J^-(\mathrm{X}_d, f)$ denotes the criterion function used to evaluate the subset obtained by removing f ($f \in \mathrm{X}_d$) from X_d. Then we shall say that $REM(\mathrm{X}_d)$ is an operation of removing feature f^- from the current set X_d to obtain set X_{d-1} if

$$REM(\mathrm{X}_d) \equiv \mathrm{X}_d \setminus \{f^-\} = \mathrm{X}_{d-1}, \qquad \mathrm{X}_d, \mathrm{X}_{d-1} \subset \mathrm{Y}.$$

In order to simplify the notation for a repeated application of FS operations we introduce the following useful notation

$$\mathrm{X}_{d+2} = ADD(\mathrm{X}_{d+1}) = ADD(ADD(\mathrm{X}_d)) = ADD^2(\mathrm{X}_d) \ ,$$
$$\mathrm{X}_{d-2} = REM(REM(\mathrm{X}_d)) = REM^2(\mathrm{X}_d) \ ,$$

and more generally

$$\mathrm{X}_{d+\delta} = ADD^\delta(\mathrm{X}_d), \qquad \mathrm{X}_{d-\delta} = REM^\delta(\mathrm{X}_d) \ .$$

Abstract class `Sequential_Step` forms a basis for concrete ADD and REM implementations, see Fig. 6. Note that in standard sequential FS methods, $J^+(\cdot)$ and $J^-(\cdot)$ stand for

$$J^+(\mathrm{X}_d, f) = J(\mathrm{X}_d \cup \{f\}), J^-(\mathrm{X}_d, f) = J(\mathrm{X}_d \setminus \{f\}) \ , \quad (2)$$

where $J(\cdot)$ is the FS criterion function to be evaluated on the subspace defined by the tested feature subset. Class `Sequential_Step_Straight` implements ADD and REM as defined in (2), see Fig. 6.

2) Basic Sequential Selection: builds up a subset of the required number d of features incrementally starting with the empty set (*bottom-up* or *forward* approach), or starting with the complete set of features and iteratively removing the most redundant features until d features retain (*top-down* or *backward* approach) [1], [59].

Sequential Forward Selection (SFS) yields subsets of 1 to d features:

$\mathrm{X}_d = ADD^d(\emptyset)$.

Sequential Backward Selection (SBS) yields subsets of d to D features:

$\mathrm{X}_d = REM^{|\mathrm{Y}|-d}(\mathrm{Y})$.

Unlike BIF, the basic sequential selection takes into account feature dependencies. However, due to lack of backtracking it suffers from the so-called nesting of feature subsets [1]. To better avoid local extremes in the search space more complex (but slower) methods have been defined. Class `Search_SFS` implements both SFS and SBS, see Fig. 6.

3) Sequential Floating Search: adds a 'self-controlled backtracking' to enable correction of decisions made in previous algorithm phases [37]. For each subset size the so-far best solution is remembered. The Sequential Forward Floating Selection (SFFS) procedure repeats after each forward step conditional backward steps as long as they improve solutions for the lower subset size. The Sequential Backward Floating Search (SBFS) principle is analogous.

Sequential Forward Floating Selection (SFFS) yields subsets of all sizes, unless restricted by $\Delta \in [0, D-d]$ to obtain subsets of 1 to d features only:

1) Start with $\mathrm{X}_0 = \emptyset$, $k = 0$.
2) $\mathrm{X}_{k+1} = ADD(\mathrm{X}_k)$, $k = k + 1$.
3) Repeat $\mathrm{X}_{k-1} = REM(\mathrm{X}_k)$, $k = k - 1$ as long as it improves solutions already known for the lower k.
4) If $k < d + \Delta$ go to 2.

Floating search algorithms have shown good performance in great variety of tasks and are considered to be among the most universally applicable tools in FS [38], [60]. Class `Search_SFFS` implements both SFFS and SBFS, see Fig. 6.

4) Oscillating Search: (OS) is based on repeated modification of the current subset X_d of d features [39]. In each step a number of features is replaced by better ones until no replacement leads to improvement. The number of replaced

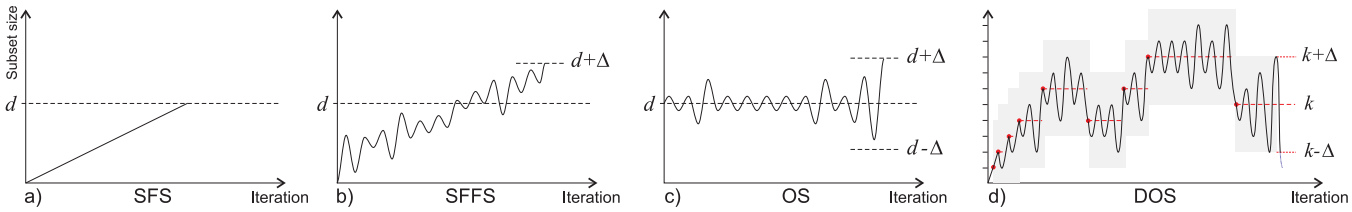

Figure 7. *Graphs demonstrate the course of d-parametrized sub-optimal sequential subset search algorithms: a) Sequential Forward Selection, b) Sequential Forward Floating Selection, c) Oscillating Search and d-optimizing algorithm: d) Dynamic Oscillating Search*

features in one step can increase from 1 up to the user-specified *limit* Δ, which enables trading accuracy for speed.

Oscillating Search (OS) yields a subset of d features, with search-extent-restricting parameter $\Delta \geq 1$:

1) Start with any initial set X_d of d features. Let $\delta = 1$.
2) Let $\mathrm{X}_d^{\downarrow} = ADD^{\delta}(REM^{\delta}(\mathrm{X}_d))$.
3) If $\mathrm{X}_d^{\downarrow}$ better than X_d, let $\mathrm{X}_d = \mathrm{X}_d^{\downarrow}$, let $\delta = 1$ and go to 2.
4) Let $\mathrm{X}_d^{\uparrow} = REM^{\delta}(ADD^{\delta}(\mathrm{X}_d))$.
5) If $\mathrm{X}_d^{\uparrow}$ better than X_d, let $\mathrm{X}_d = \mathrm{X}_d^{\uparrow}$, let $\delta = 1$ and go to 2.
6) If $\delta < \Delta$ let $\delta = \delta + 1$ and go to 2.

OS starts the search from any initial set of d features. This search scheme enables a) using OS as standalone FS method, b) tuning FS results obtained in another way, c) tackling very-high-dimensional problems (where bottom-up or top-down procedures would fail to reach d in reasonable time [61]), d) repeated starts from various random initial subsets to better avoid local extremes, e) time-controlled operation – OS intermediate solutions tend to improve faster in earlier search stages, pre-mature stopping is thus likely to give a reasonably good solution. Fig. 7 illustrates the differences in the course of search among OS and other sequential methods. Class `Search_OS` implements OS, see Fig. 6.

5) Dynamic Oscillating Search: (DOS) [40] is the d-optimizing counterpart to the d-parametrized OS as it allows change in intermediate solution cardinality in the course of search [40]. See Fig. 7 for illustration of the difference.

Dynamic Oscillating Search (DOS) yields a subset of optimized size k, with search-extent-restricting parameter $\Delta \geq 1$:

1) Start with $\mathrm{X}_{k=3} = ADD^3(\emptyset)$, or alternatively with any initial set $\mathrm{X}_k \subset \mathrm{Y}$. Let $\delta = 1$.
2) Compute $ADD^{\delta}(REM^{\delta}(\mathrm{X}_k))$; if any intermediate subset X_i, $i \in [k-\delta, k]$ is found better than X_k, let it become the new X_k with $k = i$, let $\delta = 1$ and restart step 2.
3) Compute $REM^{\delta}(ADD^{\delta}(\mathrm{X}_k))$; if any intermediate subset X_j, $j \in [k, k+\delta]$ is found better than X_k, let it become the new X_k with $k = j$, let $\delta = 1$ and go to 2.
4) If $\delta < \Delta$ let $\delta = \delta + 1$ and go to 2.

DOS shares the key advantages of OS, see Sect. IV-D4. Class `Search_DOS` implements DOS, see Fig. 6.

E. Tackling Excessive Search Complexity

In many tasks the trade-off between search complexity and accuracy of results is to be taken into account. For cases when the complexity of search appears prohibitive while the task to be solved does not permit simplification imposed by BIF style of FS (i.e., complex dependencies among features are to be expected and too important to ignore), the FST3 library enables several possible workarounds.

1) Exploiting the Oscillating principle: Especially in very-high dimensional tasks where d is neither close to 0 neither to D it may be feasible to apply OS (cf. Sect. IV-D4) or DOS (cf. Sect IV-D5) in the least computationally demanding setting ($\Delta = 1$), initialized randomly or by means of feature ranking (BIF). Unlike other sequential algorithms the oscillating algorithms by-pass the possibly time-prohibitive (or numerically unstable) process of reaching d in top-down or bottom-up manner. This approach has been shown feasible in text categorization [61].

2) Parallelization: As alternative to standard sequential evaluation of feature subset candidates FST3 enables threaded candidate subset evaluation. All FST3 sequential search methods can be easily parallelized by using `Sequential_Step_Straight_Threaded` instead of `Sequential_Step_Straight` evaluator object. The actual search speed gain depends on particular problem setting. In small-sample, low-dimensional settings, or when criterion evaluation is very fast, the actual gain can remain negligible or even negative due to thread management overhead. However, in computationally more complex cases (high dimensionality, complex criterion functions used, large sample size, etc.) the gain can become substantial. With many higher-dimensional problems the FST3 threading capability becomes key in making the feature selection task tractable. Note that maximum permitted number of threads to run in parallel is to be user-specified in `Sequential_Step_Straight_Threaded` depending on hardware capabilities.

3) Hybridization: In case the chosen FS criterion proves too slow to permit complex search schemes where large numbers of candidate subsets are to be evaluated, it may be feasible to introduce another – fast but less accurate – FS criterion to pre-filter some of the candidate subsets in each search step. Only the more promising candidates are eventually evaluated by the primary slow FS criterion.

```
#datafile
#title Medical data
; 2-class 33-dimensional data representing tissue samples
; 128 samples of benign tissue, 222 samples of malignant tissue
#features       33
#classes        2       128,222
#data
13.54  14.36  87.46  566.3  0.09779  0.08129
0.06664  0.04781  0.1885  0.05766  0.2699  0.7886
2.058  23.56  0.008462  0.0146  0.02387  0.01315
                              ⋮
```

Figure 8. Default FST3 data file format consists of textual header (`;` marks comments) followed by C-style sequence of textual numerical values

This *hybrid* approach to FS has been shown capable of yielding results comparable or only marginally different from standard single-criterion FS [4], [31]. The common option is to combine *filters* and *wrappers* (see Sec. III-A). For sake of simplicity let $J_F(.)$ denote the faster but for the given problem possibly less appropriate *filter* criterion, $J_W(.)$ denote the slower but more appropriate *wrapper* criterion. The *hybridization coefficient*, defining the proportion of feature subset evaluations to be accomplished by wrapper means, is denoted by $\lambda \in [0, 1]$. In the following $\lceil \cdot \rceil$ denotes value rounding. To implement hybridization we redefine operations ADD and REM (see Sect. IV-D1):

Definition 3: For a given current feature set X_d and given $\lambda \in [0, 1]$, let Z^+ be the set of candidate features

$$\mathrm{Z}^+ = \{f_i : f_i \in \mathrm{Y} \setminus \mathrm{X}_d; i = 1, \ldots, \max\{1, \lceil \lambda \cdot |\mathrm{Y} \setminus \mathrm{X}_d| \rceil\}\}$$

such that

$$\forall f, g \in \mathrm{Y} \setminus \mathrm{X}_d, f \in \mathrm{Z}^+, g \notin \mathrm{Z}^+ \qquad J_F^+(\mathrm{X}_d, f) \geq J_F^+(\mathrm{X}_d, g)\ ,$$

where $J_F^+(\mathrm{X}_d, f)$ denotes the pre-filtering criterion function used to evaluate the subset obtained by adding f ($f \in \mathrm{Y} \setminus \mathrm{X}_d$) to X_d. Let f^+ be the feature such that

$$f^+ = \arg\max_{f \in \mathrm{Z}^+} J_W^+(\mathrm{X}_d, f)\ ,$$

where $J_W^+(\mathrm{X}_d, f)$ denotes the main criterion function used to evaluate the subset obtained by adding f ($f \in \mathrm{Z}^+$) to X_d. Then we shall say that $ADD_H(\mathrm{X}_d)$ is an operation of adding feature f^+ to the current set X_d to obtain X_{d+1} if

$$ADD_H(\mathrm{X}_d) \equiv \mathrm{X}_d \cup \{f^+\} = \mathrm{X}_{d+1}, \qquad \mathrm{X}_d, \mathrm{X}_{d+1} \subset \mathrm{Y}.$$

Definition of REM_H is analogous. Any sequential search algorithm in FST3 can be hybridized by means of the `Sequential_Step_Hybrid` specialization of class *`Sequential_Step`*, see Fig. 6 and Sect. IV-D1 and V.

F. Improving Generalization in Overfitting-prone Scenarios

If the FS process produces poor results due to overfitting as described in Sect. III-E, FST3 provides several workarounds that may help to find better alternative solutions without necessity to resort to weaker search methods like BIF. (In BIF possible dependencies among features are completely ignored what has been shown less harming than dependencies being wrongly estimated [7], [62]–[64].)

1) Criteria Ensembles: Substituting `Sequential_Step_Ensemble` for `Sequential_Step_Straight` in sequential search algorithms enables multiple criterion functions to be used when evaluating candidate feature subsets. In one sequential search step then each of the various employed criteria builds a list of feature candidates, ordered descending according to criterion value. Feature candidate positions in all lists are then joined (averaged) and the candidate with best average position (i.e., most universal preference) is selected for addition/removal to/from the current working subset. Multiple criteria voting produces results that are more stable [65] and less affected by possibly misleading single criterion properties (analogy to overfitting).

2) Result Regularization and Result Equivalence Identification: FST3 enables tracking of all/part-of intermediate solutions that the search algorithms evaluate in the course of search. Objects derived from *`Result_Tracker`* can be linked to any search algorithm in order to enable eventual selection of alternative solutions to the single one provided by standard search. `Result_Tracker_Dupless` can reveal alternative feature subsets yielding the same criterion value. `Result_Tracker_Regularizer` makes it possible to select alternative solutions based on any secondary criterion. This option enables selecting alternative solutions that may be less likely to over-fit [6], [53].

3) Stability and Similarity Evaluation: The result tracking mechanism can be alternatively utilized to evaluate FS process stability as well as to compare output of various FS processes (see Sect. III-E). `Result_Tracker_Stability_Evaluator` collects selected subsets from various FS runs to eventually evaluate various stability and similarity measures [56].

G. Feature Acquisition Cost Minimization

FST3 enables feature acquisition cost minimization taking use of `Result_Tracker_Regularizer` in combination with `Criterion_Sum_Of_Weights` secondary criterion. The trade-off between the maximum achieved value of the primary criterion and maximal possible decrease of the sum of pre-specified feature weights is controlled by user-specified threshold. See [53] for details.

V. FST3 Library Code Usage

To select features for a given task the user needs to choose one of the implemented search methods and the appropriate criterion, both with respect to the particular problem. For recommendation see [1], [4], [5], [66]. From the library usage perspective this means setting up and linking objects of these parent types: *`Data_Accessor`* as interface to data, *`Data_Scaler`* to enable data pre-processing or normalization, one or more *`Data_Splitter`*s to specify how and what part of data to use for what purpose (learning, testing, cross-validation, etc.), `Subset` to store FS results, *`Criterion`* to

enable evaluation of candidate feature subsets in the course of search, `Search` to actually run the FS process.

Criterion can be instantiated as one of the model-based independent functions (based on `Criterion_Normal` or `Criterion_Multinomial`) or as instance of one of available `Classifier`s wrapped by `Criterion_Wrapper` object (see Figs. 1 and 5). If the `Search` object represents a sequential search algorithm, it needs an instance of the supporting `Sequential_Step` object (see Fig. 9).

In Fig. 9 we give an example of code usage. It selects features using SFFS algorithm and 3-NN wrapper classification accuracy as FS criterion. Classification accuracy (i.e, FS wrapper criterion value) is estimated on the first 50% of data samples by means of 3-fold cross-validation. The final classification performance on the selected subspace is eventually validated on the second 50% of data. SFFS is called in d-optimizing setting, invoked by parameter 0 in `search(0,...)`, which is otherwise used to specify the required subset size. For more examples see http://fst.utia.cz/?fst3_usage.

A. FST3 Library Applications

FST3 has served as research and application platform in wide range of domains. It has been successfully applied in credit scoring [67], medical diagnostics, economics (evaluation of business success factors), text categorization [61] and various specialized recognition tasks. Its search algorithms have been applied beyond the boundaries of feature selection for texture synthesis in automotive industry [68].

Provided the training data is available in or convertible to ARFF (Weka) or TRN format or a problem-specific data accessor object can be implemented, the library enables solving various FS problem scenarios. Several of the included tools provide functionality beyond the limits of concurrent libraries. Oscillating Search (cf. Sect. IV-D4) enables non-trivial FS in very-high-dimensional problems, multinomial Bhattacharyya distance (cf. Sect. IV-C3 and IV-E1) is an unconventional FS criterion practical in document processing, Dynamic Oscillating Search is highly efficient in feature subset size optimizing scenarios (cf. Sect. IV-D5), etc.

VI. Alternative Libraries

Other implementations of FS methods in Matlab can be found at http://www.prtools.org, at http://cmp.felk.cvut.cz/cmp/software/stprtool, in C/C++ at http://pcp.sourceforge.net, at http://www.sgi.com/tech/mlc, at http://sites.google.com/site/tooldiag, or at http://www.cs.waikato.ac.nz/ml/weka. For a comprehensive list of alternative FS related projects as well as other resources including benchmarking data see http://fst.utia.cz/?relres.

VII. Summary and Future Work

We provide an advanced C++ library for feature (attribute, variable) selection in machine learning. The library, available free for non-commercial use at http://fst.utia.cz, collects a number of data modeling, subset evaluation and subset search tools. The particular advantage over concurrent solutions is its clear FS-focused architecture, allowing for implementing various non-trivial search scenarios, useful to tackle the computational complexity vs. result accuracy trade-off. The library forms a basis for further development, allowing extension in each of its main functional areas (co-operation welcome). The next releases will include optimal search methods, in particular those based on the Branch & Bound idea.

Acknowledgment

The work has been supported by grants of the Czech Ministry of Education 1M0572 DAR and 2C06019 ZIMOLEZ.

References

[1] P. A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*. Prentice Hall, 1982.

[2] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Trans. PAMI*, vol. 22, no. 1, pp. 4–37, 2000.

[3] S. Mitra *et al.*, "Data mining in soft computing framework: A survey," *IEEE Trans. NN*, vol. 13, no. 1, pp. 3–14, 2002.

[4] H. Liu and L. Yu, "Toward integrating FS algorithms for classification and clustering," *IEEE Trans. on KDE*, vol. 17, no. 4, pp. 491–502, 2005.

[5] R. Kohavi and G. H. John, "Wrappers for feat. subs. sel." *Artif. Intell.*, vol. 97, no. 1-2, pp. 273–324, 1997.

[6] Š. J. Raudys, "Feature over-selection," in *Proc. S+SSPR*, vol. LNCS 4109. Springer, 2006, pp. 622–631.

[7] L. I. Kuncheva, "A stability index for FS," in *Proc. 25th IASTED Int. Mul.-Conf. AIAP'07*. ACTA Pr., 2007, pp. 390–395.

[8] M. Ben-Bassat, "Pattern recognition and reduction of dimensionality," *Handbook of Statistics*, vol. 2, pp. 773–910, 1982.

[9] R. Gilad-Bachrach, A. Navot, and N. Tishby, "Margin based FS – theory and algorithms," in *ICML '04: Proc. 21th Int. Conf. on Machine Learning*. ACM Press, 2004, p. 43.

[10] Y. Sun, "Iterative relief for feature weighting: Algorithms, theories, and applications," *IEEE Trans. PAMI*, vol. 29, no. 6, pp. 1035–1051, 2007.

[11] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Int. Conf. on Machine Learning*, 1996, pp. 148–156. [Online]. Available: www.boosting.org

[12] S. Z. Li, Z. Zhang, H. Shum, and H. Zhang, "Floatboost learning for classification," in *Proc. 16th Ann. Conf. on NIPS*, no. 15. MIT Press, 2002.

[13] J. Zhu, H. Zou, S. Rosset, and T. Hastie, "Multiclass adaboost," Department of Statistics, University of Michigan, Tech. Rep. Technical Report No. 430, 2005.

```
void Feature_Selection_Wrapper_kNN()
{
	typedef double RETURNTYPE;  typedef double DATATYPE;  typedef double REALTYPE;
	typedef unsigned int IDXTYPE;  typedef unsigned int DIMTYPE;  typedef short BINTYPE;
	typedef FST::Subset<BINTYPE, DIMTYPE> SUBSET;
	typedef FST::Data_Intervaller<vector<FST::Data_Interval<IDXTYPE> >,IDXTYPE> INTERVALLER;
	typedef FST::Data_Splitter_CV<INTERVALLER,IDXTYPE> SPLITTERCV;
	typedef FST::Data_Splitter_5050<INTERVALLER,IDXTYPE> SPLITTER5050;
	typedef FST::Data_Accessor_Mem_TRN<DATATYPE,IDXTYPE,INTERVALLER> DATAACCESSOR;
	typedef FST::Distance_Euclid<DATATYPE,DIMTYPE,SUBSET> DISTANCE;
	typedef FST::Classifier_kNN<RETURNTYPE,DATATYPE,IDXTYPE,DIMTYPE,SUBSET,DATAACCESSOR,DISTANCE> CLASSIFIERKNN;
	typedef FST::Criterion_Wrapper<RETURNTYPE,SUBSET,CLASSIFIERKNN,DATAACCESSOR> WRAPPERKNN;
	typedef FST::Sequential_Step_Straight<RETURNTYPE,DIMTYPE,SUBSET,WRAPPERKNN> EVALUATOR;
	typedef boost::shared_ptr<FST::Data_Splitter<INTERVALLER,IDXTYPE> > PSPLITTER;

	// keep second half of data for independent testing of final classification performance
		PSPLITTER dsp_outer(new SPLITTER5050());
	// in the course of search use the first half of data by 3-fold cross-validation in wrapper FS criterion evaluation
		PSPLITTER dsp_inner(new SPLITTERCV(3));
	// do not scale data
		boost::shared_ptr<FST::Data_Scaler<DATATYPE> > dsc(new FST::Data_Scaler_void<DATATYPE>());
	// set-up data access
		boost::shared_ptr<vector<PSPLITTER> > splitters(new vector<PSPLITTER>);
		splitters->push_back(dsp_outer); splitters->push_back(dsp_inner);
		boost::shared_ptr<DATAACCESSOR> da(new DATAACCESSOR("data/speech.trn",splitters,dsc));
		da->initialize();
	// initiate access to split data parts
		da->setNestingDepth(0); if(!da->getFirstSplit()) throw FST::fst_error("50/50 data split failed.");
		da->setNestingDepth(1); if(!da->getFirstSplit()) throw FST::fst_error("3-fold cross-validation failure.");
	// initiate the storage for subset to-be-selected
		boost::shared_ptr<SUBSET> sub(new SUBSET(da->getNoOfFeatures()));  sub->deselect_all();
	// set-up 3-Nearest Neighbor classifier based on Euclidean distances
		boost::shared_ptr<CLASSIFIERKNN> cknn(new CLASSIFIERKNN); cknn->set_k(3);
	// wrap the 3-NN classifier to enable its usage as FS criterion (criterion value will be estimated by 3-fold cross-val.)
		boost::shared_ptr<WRAPPERKNN> wknn(new WRAPPERKNN);
		wknn->initialize(cknn,da);
	// set-up the standard sequential search step object (option: hybrid)
		boost::shared_ptr<EVALUATOR> eval(new EVALUATOR);
	// set-up Sequential Forward Floating Selection search procedure
		FST::Search_SFFS<RETURNTYPE,DIMTYPE,SUBSET,WRAPPERKNN,EVALUATOR> srch(eval);
		srch.set_search_direction(FORWARD);
	// run the search
		RETURNTYPE critval_train, critval_test;
		if(!srch.search(0,critval_train,sub,wknn)) throw FST::fst_error("Search not finished.");
	// (optionally) validate result by estimating kNN accuracy on selected feature sub-space on independent test data
		da->setNestingDepth(0);
		cknn->train(da,sub);
		cknn->test(critval_test,da);
}
```

Figure 9. FST3 library usage example – selecting features using SFFS method and 3-NN wrapper classification accuracy as FS criterion

[14] D. Redpath and K. Lebart, "Observations on boosting FS," in *Proc. MCS 2005 : Multiple Classifier Systems (Seaside, CA)*, vol. LNCS 3541. Springer, 2006, pp. 32–41.

[15] R. Nock and F. Nielsen, "A real generalization of discrete adaboost," *Artif. Intell.*, vol. 171, no. 1, pp. 25–41, 2007.

[16] G. Brown, "A New Perspective for Information Theoretic Feature Selection," in *Proc. AISTATS '09, vol. 5 of JMLR: W&CP 5*, 2009, pp. 49–56.

[17] H. Peng *et al.*, "FS based on mutual inf.: Criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Trans. PAMI*, vol. 27, no. 8, pp. 1226–1238, 2005.

[18] K. Kryszczuk and A. Drygajlo, "Impact of feature correlations on separation between bivariate normal distributions," in *ICPR 2008*. IEEE Comp. Soc., 2008.

[19] M. A. Hall, "Correlation-based FS for discrete and numeric class machine learning," in *Proc. 17th ICML*. Morgan Kaufmann, 2000, pp. 359–366.

[20] J. Biesiada and W. Duch, "FS for high-dimensional data: A kolmogorov-smirnov correlation-based filter," in *Proc. CORES, 4th Int. Conf. on Comp. Rec. Syst.*, 2005.

[21] H. Almuallim and T. Dietterich, "Learning boolean concepts in the presence of many irrelevant features," *Artificial Intelligence*, vol. 69, no. 1-2, pp. 279–305, 1994.

[22] A. Arauzo-Azofra, J. M. Benítez *et al.*, "C-focus: A continuous extension of focus," in *Proc. 7th Online W.Conf. on Soft Comp. in Indust. Apps.*, 2003, pp. 225–232.

[23] C. Hallum, "FS via an upper bound (to any degree tightness) on probability of misclassification," in *Proc. of the Conf. on Machine Processing of Remotely Sensed Data*. Purdue University, West Lafayette, Indiana, 1973.

[24] C. H. Chen, "On information and distance measures, error bounds, and FS," *Inform. Sciences*, vol. 10, pp. 159–171, 1976.

[25] D. E. Boekee *et al.*, "Some aspects of error bounds in FS," *Patt. Rec.*, vol. 11, pp. 353–360, 1979.

[26] M. Ben-Bassat, "On the sensitivity of the probability of error rule for feature selection," *IEEE Trans. Pattern Anal. Mach. Intell. PAMI*, vol. 2, pp. 57–60, 1980.

[27] H. Avi-Itzhak and T. Diep, "Arbitrarily tight upper and lower bounds on the bayesian probability of error," *IEEE Trans. PAMI*, vol. 18, no. 1, pp. 89–91, 1996.

[28] L. Devroye, L. Györfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*. Springer, 1996.

[29] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Int. Conf. on Mach. Learning*, 1994, pp. 121–129.

[30] M. Sebban and R. Nock, "A hybrid filter/wrapper approach of FS using information theory," *Pattern Recognition*, vol. 35, pp. 835–846, 2002.

[31] P. Somol, J. Novovičová, and P. Pudil, "Flexible-hybrid sequential floating search in statistical FS," in *Proc. S+SSPR, LNCS 4109*. Springer, 2006, pp. 632–639.

[32] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, Eds., *Feature Extraction – Foundations and Applications*, ser. Studies in Fuzziness and Soft Comp. Physica, Springer, 2006, vol. 207.

[33] J. Novovičová, P. Pudil, and J. Kittler, "Divergence based FS for multimodal class densities," *IEEE Trans. PAMI*, vol. 18, no. 2, pp. 218–223, 1996.

[34] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. PAMI*, vol. 20, no. 8, pp. 832–844, 1998.

[35] P. Somol, P. Pudil, and J. Kittler, "Fast branch & bound algorithms for optimal FS," *IEEE Trans. on PAMI*, vol. 26, no. 7, pp. 900–912, 2004.

[36] S. Nakariyakul and D. P. Casasent, "Adaptive branch and bound algorithm for selecting optimal features," *Pattern Recogn. Lett.*, vol. 28, no. 12, pp. 1415–1427, 2007.

[37] P. Pudil, J. Novovičová, and J. Kittler, "Floating search methods in FS," *Patt. Rec. Lett.*, vol. 15, no. 11, pp. 1119–1125, 1994.

[38] S. Nakariyakul and D. P. Casasent, "An improvement on floating search algorithms for feat. subs. sel." *Pattern Recognition*, vol. 42, no. 9, pp. 1932–1940, 2009.

[39] P. Somol and P. Pudil, "Oscillating search algorithms for FS," in *ICPR 2000*, vol. 02. IEEE Comp. Sc., 2000, pp. 406–409.

[40] P. Somol, J. Novovičová, J. Grim, and P. Pudil, "Dynamic oscillating search algorithms for FS," in *ICPR 2008*. IEEE Comp. Soc., 2008.

[41] C. Lai, M. J. T. Reinders, and L. Wessels, "Random subspace method for multivariate FS," *Pattern Recogn. Lett.*, vol. 27, no. 10, pp. 1067–1076, 2006.

[42] F. Hussein, R. Ward, and N. Kharma, "Genetic alg. for FS and weighting, a review and study," in *Proc. 6th ICDAR*, vol. 00. IEEE Comp. Soc., 2001, pp. 1240–1244.

[43] Z. Zhu, Y. Ong, and M. Dash, "Wrapper-filter feature selection algorithm using a memetic framework." *IEEE Trans. on Syst., Man, and Cyb., Part B*, vol. 37, no. 1, p. 70, 2007.

[44] R. Jensen, *Performing FS with ACO*, ser. Studies in Computational Intelligence. Springer, 2006, vol. 34, pp. 45–73.

[45] F. W. Glover and G. A. Kochenberger, Eds., *Handbook of Metaheuristics*, ser. Int. Ser. in Operat. Research & Management Science. Springer, 2003, vol. 57.

[46] M.-A. Tahir *et al.*, "Simultaneous FS and feature weighting using hybrid tabu search/k-nearest neighbor classifier," *Patt. Rec. Lett.*, vol. 28, no. 4, pp. 438–446, 2007.

[47] I. Gheyas and L. Smith, "Feat. sub. sel. in large dimens. domains," *Pattern Recognition*, vol. 43, no. 1, pp. 5–13, 2010.

[48] Y. Yang and J. O. Pedersen, "A comparative study on FS in text categorization," in *ICML '97: Proc. 14th Int. Conf. on Machine Learning*. Morgan Kaufmann, 1997, pp. 412–420.

[49] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comp. Surveys*, vol. 34, no. 1, pp. 1–47, 2002.

[50] E. P. Xing, *FS in Microarray Analysis*. Springer, 2003, pp. 110–129.

[51] Y. Saeys, I. naki Inza, and P. L. naga, "A review of FS techniques in bioinformatics," *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[52] I. Guyon and A. Elisseeff, "An introduction to variable and FS," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.

[53] P. Somol, J. Grim, and P. Pudil, "The problem of fragile feat. subset preference in FS methods and a proposal of algorithmic workaround," in *ICPR 2010*. IEEE Comp. Sc., 2010.

[54] P. Paclík, R. P. W. Duin, G. M. P. van Kempen, and R. Kohlus, "On FS with measurement cost and grouped features," in *Proc. S+SSPR*. Springer, 2002, pp. 461–469.

[55] A. Kalousis, J. Prados, and M. Hilario, "Stability of FS algorithms: A study on high-dimensional spaces," *Knowledge and Information Systems*, vol. 12, no. 1, pp. 95–116, 2007.

[56] P. Somol and J. Novovičová, "Evaluating stability and comparing output of feature selectors that optimize feature subset cardinality," *IEEE Transactions on PAMI*, vol. 32, pp. 1921–1939, 2010.

[57] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

[58] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for SVM*, 2001, http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[59] A. W. Whitney, "A direct method of nonparametric measurement selection," *IEEE Trans. Comput.*, vol. 20, no. 9, pp. 1100–1103, 1971.

[60] P. Somol, P. Pudil, J. Novovičová, and P. Paclík, "Adaptive floating search methods in FS," *Pattern Recogn. Lett.*, vol. 20, no. 11-13, pp. 1157–1163, 1999.

[61] J. Novovičová, P. Somol, and P. Pudil, "Oscillating feature subset search algorithm for text categorization," in *Proc. S+SSPR*, vol. LNCS 4109. Springer, 2006, pp. 578–587.

[62] J. Reunanen, "Overfitting in making comparisons between variable selection methods," *J. Mach. Learn. Res.*, vol. 3, pp. 1371–1382, 2003.

[63] ——, "A pitfall in determining the optimal feature subset size," in *Proc. 4th Int. Workshop on Pat. Rec. in Inf. Systs (PRIS 2004)*, 2004, pp. 176–185.

[64] ——, "Less biased measurement of FS benefits," in *Stat. and Optimiz. Perspectives Workshop, SLSFS*, vol. LNCS 3940. Springer, 2006, pp. 198–208.

[65] P. Somol, J. Grim, and P. Pudil, "Criteria ensembles in FS," in *Proc. MCS, LNCS 5519*. Springer, 2009, pp. 304–313.

[66] P. Somol, J. Novovičová, and P. Pudil, *Efficient Feature Subset Selection and Subset Size Optimization*. INTECH, 2010, pp. 75–97. [Online]. Available: http://www.sciyo.com/books/show/title/pattern-recognition-recent-advances

[67] P. Somol, B. Baesens, P. Pudil, and J. Vanthienen, "Filter- versus wrapper-based feature selection for credit scoring," *Int. J. of Intelligent Systems*, vol. 20, no. 10, pp. 985–1000, 2006.

[68] R. Klein, J. Meseth *et al.*, "Realreflect – real-time visualization of complex reflectance behaviour in virtual protoyping," in *Eurographics Industrial and Project Presentations*, 2003.