Proposal and examination of the FLAP algorithm

Daniel Giterman* and Eyal Brill

Faculty of Technology Management,
Holon Institute of Technology,
5810201 Holon, Israel
Email: dgdg18@gmail.com
Email: eyalb@hit.ac.il
*Corresponding author

Abstract: In real classification problems, common learning algorithms generally fail to describe instances that require complicated classification logic. Additionally, it is often difficult to ensure a satisfying amount of classified data for their training. In this work, we propose and examine a new learning algorithm that also integrates expert logic. Essentially, this algorithm takes advantage of unclassified data to produce a self-generated fuzzy inference system that is eventually used as a classifier. It also utilises a mere sample of classified data in order to compare various classifiers constructed from different algorithm options, thus finally achieving an assumingly more accurate result. As part of our study, this algorithm was compared with six well-known supervised learning algorithms such as artificial neural networks, support vector machine and random forest. We used the ten-fold cross-validation technique with Kappa statistic to assess algorithm performance. Subsequently, in order to find statistically significant dissimilarities among the algorithms, we used a two-tailed Friedman test. After the null hypothesis was rejected, we used a Nemenyi post-hoc test to prove differences between pairs of algorithms. Consequently, despite lacking in efficiency and scalability, our algorithm proved to be highly competitive and demonstrated excellent classification potential.

Keywords: fuzzy logic; fuzzy inference systems; learning algorithms; hybrid algorithms; algorithms comparison.


Biographical notes: Daniel Giterman holds a BSc degree in Applied Mathematics and an MSc degree in Management of Technology from Holon Institute of Technology. He currently works as a data scientist, while focusing on analysis in the fields of e-commerce and online marketing. His main professional interests are in machine learning, statistical data analysis, and statistical computing and modelling.

Eyal Brill holds a PhD degree in Agr. Economics from the Hebrew University, Jerusalem. Currently, he is the Head of BSc program, Faculty of Technology Management. His main professional interest is in machine learning and abnormality detection. He is also the Chair of Working Group 9 of ISO Committee 224, which is responsible for the development of ISO Standard 24510 – ‘Water quality abnormality detection’.

Copyright © 2018 Inderscience Enterprises Ltd.
1 Introduction

Machine learning is considered to be a rapidly developing field, since nowadays there are many algorithms that offer fitting solutions for obtaining insights from data. In many cases and especially in supervised learning, these algorithms perform a training procedure on already-classified data, which ends in creating a function or mathematical model that ultimately describes the data, and can be used later for classifying other data in similar structures. Many learning algorithms differ from each other and can be both naïve and very complex. Thus, deciding a priori on a specific algorithm that would describe data with minimal error is likely to be very difficult. For that reason, various statistical comparison methods have been proposed. These methods can determine – with high probability – which algorithm performs best on a particular piece of data. Learning algorithms, once they are correctly selected, can greatly benefit data analysis and support decision making in businesses and numerous industries. However, in the practical world, there is a major problem – the amount of classified data is often very low. Therefore, these algorithms train on less data and thus have a theoretically lower chance of correctly classifying the required data. Moreover, in practical classification problems, complicated elements can sometimes be easily neglected by common learning algorithms. These classification elements are generally related to a certain human logic or decision mechanism; for example, classification of a rare disease by analysing medical tests, classification of a particular mental state of a person or the classification of a security check for potential hidden weapons. In many such cases, there are conditions that are difficult to formulate. Even though it is possible to find algorithms in the scientific literature that somewhat combine expert logic and perhaps also suggest plausible solutions for such practical problems, there is no single algorithm that is considered to be the most common in practice.

This paper proposes and examines a new learning algorithm called fuzzy logic analytical package (FLAP) – an experimental algorithm created by the authors. FLAP was originally made to solve and simplify practical decision-making problems by classifying needed data. More precisely, it is a composite algorithm that utilises various models and methods in order to automatically construct a fuzzy inference system, which is then used as a classifier. Essentially, FLAP applies different methods on partially classified training data and uses automatically created logical rules in order to eventually classify the required test data. The algorithm basically combines a mechanism of machine learning with expert logic, assuming that this combination could improve the precision of data classification. The part of the algorithm that performs the learning is called FLAP-SLM (self-learning module); due to its importance, it receives separate attention in this paper, although it is definitely an integral part of FLAP.

As was mentioned earlier, FLAP is a new algorithm whose performance has yet to be tested, especially in comparison to other well-known learning algorithms such as artificial neural network (ANN), support vector machine (SVM) and random forest. Additionally, FLAP has not been checked regarding its sensitivity to input variables, and there have been no statistical comparison tests done with legitimate validation methods, such as cross-validation. Therefore, a thorough examination was conducted in order to investigate the overall efficiency and performance of this proposed algorithm.
2 Related work and methods

2.1 Semi-supervised learning

Semi-supervised learning is a sub-group of supervised learning that also relates to unclassified data. This type of learning method is very useful in practical data classification problems, as unclassified data is, in many cases, more common while classified data is considered to be a scarce resource. This type of situation is not surprising because it takes time and significant ability from human experts to classify large amounts of data without losing precision and consistency during the process. Moreover, such a process could also be very expensive; thus, semi-supervised learning algorithms try to tackle this kind of situation. Broadly speaking, semi-supervised learning algorithms try to use both classified and unclassified data in order to surpass the classification performance that could be achieved from dumping the unclassified data and using only a supervised learning method, or from dumping the classified data and using only an unsupervised learning method. It is possible to find different semi-supervised learning methods in the scientific literature (Prakash and Nithya, 2014). Among others, an interesting approach that includes integration with meta learning is often considered to enhance classification precision. The literature offers a number of hybrid algorithms that combine these learning methods, for example the semiboost algorithm (Mallapragada et al., 2007).

2.2 Meta-learning

Meta learning algorithms activate other learning algorithms such as decision trees, which are often more naïve, and apply a learning procedure to their results. In many cases, meta learning algorithms compute the final classifications from fixing the results of their classifiers, rather than simply choosing the best classifier. Generally, meta learning methods try to dynamically improve the bias by finding a suitable learning strategy according to the experience and knowledge acquired from other learning activities (Maurer, 2005). Hence, these algorithms seem to exploit the given data in a significantly better way. This characterisation is quite common in the scientific literature, but it turns out that the application itself greatly varies from one researcher to another. It is possible to find multiple applications of meta learning algorithms, for example in WEKA software, and additionally some comparative performance tests conducted on a few of them (Vijayarani and Muthulakshmi, 2013). It is also important to mention that meta learning algorithms mainly differ in their voting mechanism, which indicates the simplest method of combining results from multiple classifiers. Two of the most common voting mechanisms are bagging and boosting, and they are described in detail in Bauer and Kohavi (1998). However, it is important to note that Freund and Schapire (1999) suggested a more advanced development of boosting, which they called adaptive boosting (AdaBoost). Their algorithm basically improves the performance of multiple weak algorithms, whose error is capped at 50%, since they still have to be better than random guessing (Mehr, 2013). Usually the boosting method reaches a higher precision level than bagging; yet, it is more sensitive to the nature of the data and especially to outliers that can even cause the method to fail (Patel et al., 2013).
2.3 Hybrid algorithms

Meta learning algorithms can be applied in many different ways and are also useful in multiple kinds of combinations, whose main purpose is to improve classification precision and algorithm flexibility. We shall focus on an interesting combination of meta learning and artificial intelligence, whose primary goal is to offer an alternative for algorithms that struggle to handle uncertainty, noisy data or complicated classification logic. Recently, artificial intelligence has become quite a popular field among researchers, for example ANNs (Abraham, 2005); swarm algorithms and their combination with neural networks and fuzzy logic (Abraham et al., 2006); and neuro-fuzzy algorithms, which combine and exploit both the learning abilities of neural networks and the decision characteristics of fuzzy logic (Babuška and Verbruggen, 2003). Another interesting addition is the optimisation of such algorithms by integrating genetic algorithms (Ferariu et al., 2005).

2.4 Fuzzy logic

Fuzzy logic is a mathematical theory that includes and expands the classical binary logic. It is based on the fuzzy set theory, where the notion of a binary set membership has been altered to contain partial membership ranging between 0 and 1 (Zadeh, 1965). Unlike their crisp counterparts, fuzzy sets have ambiguous boundaries which enable gradual transitions between defined sets. This fact allows for uncertainty associated with these concepts to be modelled directly. Fuzzy logic was basically meant to deal with imprecise, uncertain, ambiguous or partially true data. In the practical world, such data is usually not an exception, but the norm. Therefore, fuzzy logic is used in numerous applications, such as the complex control of non-linear systems, expert systems, pattern recognition and more. Generally, the practical implementation is manifested via a fuzzy inference system, which is based on the principles of fuzzy logic and consists of several essential parts.

**Figure 1** Fuzzy inference system

In fuzzy inference systems, fuzzy sets are depicted by membership functions. These functions are generally represented by a number of geometric shapes, the most common being the triangle and trapeze, due to their simplicity (Pal and Shiu, 2004). Other shapes may include the Gaussian, sigmoid, S and bell shape. In order to create the membership functions, a number of methods were proposed (Bouchon-Meunier et al., 1996).
Accordingly, automatic methods are the most appropriate for fuzzy inference systems that strive for full automation. A good example of this is described in Jamsandekar and Mudholkar (2014), where the authors presented an automatic process for creating membership functions by using a clustering model. Moreover, the authors compared the results of this method with human experts, and it showed a relatively high match. Hence, automatic methods are able to generate close results to those of experts’ knowledge methods. Eventually, membership functions are regarded as a means of value conversion, while the logic of mapping inputs to outputs is hidden within the fuzzy rules. These rules are built in the form of ‘if-then’, and refer to the membership functions of both inputs and outputs. In many cases, in order to simplify the logic description, researchers and practitioners use the ‘and’ operator when there are multiple variables. Thus, the rules are very similar to the following – if $x_1$ is $A_1$ and … $x_n$ is $A_n$ then $y$ is $B$. Yet, not only for the sake of simplicity, the ‘and’ operator is used, due to its good coverage of different situations, including those that are even more appropriately defined by other operators such as ‘or’, ‘not’ and ‘unless’ (Chen and Pham, 2001). Another important part of the fuzzy inference system is the inference engine, which is used to process the rules in parallel via a fuzzy inference procedure, such as the max-min or max-product operations (Jang et al., 1997). Additionally, it is highly relevant to note that there are two main types of fuzzy inference systems, Sugeno (or TSK) (Takagi and Sugeno, 1985) and Mamdani (Mamdani and Assilian, 1975). In the Sugeno-type, the output is generated as a crisp value; hence, there is no need for an output membership function and a defuzzification procedure (Ross, 2004). However, in the Mamdani-type the output is computed as a fuzzy number; therefore, it has to be defuzzified in order to obtain the final crisp result. Despite being probably slower, due to additional computations, the Mamdani-type system is generally more favourable because it is more obvious and intuitive for humans.

2.5 Summary

The scientific literature offers quite a few learning algorithms that can potentially present satisfying solutions for practical data classification problems. Various learning algorithms, such as AdaBoost and ANNs and their combinations with others, are particularly of high interest. However, no algorithm that could train on a small, classified sample and handle complicated classification logic has been found to deliver substantial results. Therefore, it is important to continue the research and examination of such learning algorithms, and propose new solutions to the industry that can produce better results.

3 Methodology and experiment

3.1 Introduction to FLAP

FLAP is a classification algorithm that uses expert logic and multiple analytical methods in order to automatically construct a fuzzy inference system that will serve as a classifier. Under no circumstances does FLAP perform any learning activities on the data, since the FLAP-SLM learning module fulfils this role, as it was particularly created for this specific purpose (see Section 3.3).
According to Figure 2, the first step consists of five actions, whose ultimate purpose is to build the classifier. After that, the second step is executed; it contains four actions that basically take the constructed classifier and use it to classify the required data. Evidently, every action fulfills a certain requirement of the actions that follow; hence, the order is of great significance. In both steps, the first two actions (1.1, 1.2, 2.1 and 2.2) deal with data preparation. Most importantly, the pre-processing action splits the input variables into separate sets for further procedures. Additionally, it handles missing data, exclusion of special observations, and the activation of different modes such as data aggregation. Obviously, the latter two must be defined manually, if needed. Lastly, the other four actions (1.3, 1.4, 1.5, and 2.3) are considered the core of the algorithm, and are described in detail in Section 3.2.

### 3.2 Description of FLAP core actions

Outliers are a common phenomenon that can potentially cause high error in practical data classification problems. For this reason, Action 1.3 deals with such irregular data, which may appear in the training set. Due to the fact that input variables are handled separately throughout the construction step and their distributions in practice are rarely close to normal, FLAP uses a univariate and non-parametric outlier detection method called median absolute deviation (MAD) (Leys et al., 2013). The main purpose behind this action is to ensure that the algorithm constructs the classifier only by using significant data.

Let us consider a training set in which there are \( n \) input variables, so that \( 2 \leq n \leq 6 \) and \( j = 1, \ldots, n \). For each variable \( X^j = \{x_1, \ldots, x_\alpha\} \), \( \alpha > 0 \), the value of \( MAD_j \) is computed by the following formula:

\[
MAD_j = b \cdot \text{Median} (X^j) - \text{Median} (X^j)
\]  

(1)
The $b$ parameter is calculated from the distribution of $X_j$ so that $b = sd(X_j) / |Q_{0.75}(X_j) - \text{Median}(X_j)|$. Afterwards, for every datum $x_i (i = 1, \ldots, \alpha)$ in each variable $X_j$, the following condition is tested:

$$\frac{|x_i - \text{Median}(X_j)|}{\text{MAD}(X_j)} > \alpha, \alpha \in \mathbb{R}^+ \tag{2}$$

If inequality (2) holds, $x_i$ will be removed from $X_j$.

Action 1.4 applies a clustering model in order to determine the fuzzy sets for each input variable. Although clustering models are usually used on multivariate data, in this specific case, these models are applied on each input variable separately, hence somewhat resembling density functions. Additionally, it is important to mention that most clustering models require a predefined number of clusters. Therefore, we assumed a fixed number of clusters as a practical limitation. The assumption of five clusters per each input variable is theoretically incorrect as it is not always optimal, although greatly simplifying algorithm realisation. FLAP may employ any of the following implemented models:

- **K-Means++ (KMPP)** – introduced in Arthur and Vassilvitskii (2007) as a more advanced model than the classical K-Means. In their paper, Arthur and Vassilvitskii showed that this model provides a solution with an error of no more than $O(\log k)$ from the most optimal solution of the classical K-Means on any dataset. Additionally, unlike the classical K-Means, this model is deterministic, due to the initialisation method of the first centroids.

- **Partition around medoids (PAM)** – similarly to the classical K-Means, this model tries to iteratively minimise the sum of squared errors. However, its main difference from K-Means is expressed in the representation of the clusters. Thus, each cluster is described by the most central observation located inside the cluster, rather than by the calculated average, which may not even belong to the cluster (Kaufman and Rousseeuw, 1987). Despite requiring much greater computational effort, this model is deterministic and more robust in the presence of outliers than the classical K-Means.

- **Fuzzy C-Means (FCM)** – a deterministic model that implements K-Means in combination with fuzzy logic principles. Similar to the previous models, this model shifts the cluster centroid iteratively, until it converges. However, each observation belongs to a cluster with a particular degree of membership (Bezdek et al., 1984). Thus, an observation has a higher level of belonging in a cluster when it is closer to the centroid and vice versa.

- **Hierarchical agglomerative clustering (HAC)** – a bottom-up model that starts by assigning each observation to its’ own cluster, while a pair of the nearest clusters merges in every following iteration, until only one large cluster remains. After each union, a new distance is computed between the recently-merged cluster and the others, according to the Lance-Williams formula (Murtagh and Contreras, 2012). Eventually, this model creates a dendrogram that displays all of the clusters’ nesting stages; therefore, cutting it at a certain point delivers the requisite result.
Consider that each datum $x_i \in X'$ belongs to a cluster $g_i^j$ while $k = 1, \ldots, 5$ and $g_i^k$ is received as a result of applying a clustering model (out of the previously stated ones) on an input variable $X'$:

\[ \{g_i^1, \ldots, g_i^5\} = \begin{cases} 
\text{KMPP}(X'), & k = 5 \\
\text{PAM}(X'), & k = 5 \\
\text{FCM}(X'), & k = 5 \\
\text{HAC}(X'), & k = 5 
\end{cases} \quad (3) \]

So that for every $g_i^j = \{x_1, \ldots, x_{\alpha}\}$, when $\bar{\alpha}$ represents the index of the last datum in the cluster, a centroid is eventually computed as $c_i^j = \sum x_i \in g_i^j / \bar{\alpha}$. Therefore, each input variable $X'$ acquires $c_i^1, \ldots, c_i^5$, so that $c_i^1 < c_i^2 < c_i^3 < c_i^4 < c_i^5$.

Action 1.5 constructs a fuzzy inference system in two main phases that consist of generating membership functions and fuzzy rules, while using insights obtained from previous actions and expert knowledge.

The creation of membership functions is performed while utilising the results of Action 1.4, so that clusters $g_i^1, \ldots, g_i^5$ are regarded as the origin for fuzzy sets $f_i^1, \ldots, f_i^5$ per each input variable $X'$. Hence, it is assumed that each input variable has five membership functions that describe a specific quality rank, for example: very low, low, medium, high and very high. Thus, for instance, a relatively low value may belong to the Low group while a slightly higher value can already belong more to the medium group. The algorithm generates two kinds of membership functions – triangle and trapeze – for each input variable $X'$ as defined by the following:

\[ \mu_{f_i^k}(x_i) = \begin{cases} 
1, & x_i = c_i^k (1 \leq k \leq 5) \\
0, & x_i = c_{i,1}^k (2 \leq k \leq 4) \\
0, & x_i = c_{i,1}^k (k = 1) \\
0, & x_i = c_{i,1}^k (k = 5) 
\end{cases} \quad (4) \]

According to (4), centroids $c_i^k$ are values whose membership degree to a fuzzy set $f_i^k$ is equal to 1, while their membership degree to $f_i^k_{\alpha}$ is equal to 0 (as they are also boundaries of adjacent fuzzy sets). From a geometric perspective, it can be represented similarly to that in Figure 3.

As portrayed in Figure 3, FLAP implements type-1 membership functions (Gupta, 2014), while using triangles for middle sets and trapezes for fringes. Thus, all values belong to at least one set and have a full overall membership.
Owing to the construction of a Mamdani-type fuzzy inference system, membership functions are also created for the output variable. By default, these functions are created in a similar manner for nine fuzzy sets in a [0, 100] range, while all sets are equal. However, it is possible to change the amount and ranges if needed.

**Table 1**  Description of output variable fuzzy sets

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Fuzzy set range</th>
<th>Centroid</th>
<th>Intercept</th>
<th>Fuzzy rules range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Poor</td>
<td>[0–20]</td>
<td>10</td>
<td>15</td>
<td>[0–15]</td>
</tr>
<tr>
<td>3</td>
<td>Low</td>
<td>[20–40]</td>
<td>30</td>
<td>35</td>
<td>(25–35)</td>
</tr>
<tr>
<td>4</td>
<td>Almost medium</td>
<td>[30–50]</td>
<td>40</td>
<td>45</td>
<td>(35–45)</td>
</tr>
<tr>
<td>5</td>
<td>Medium</td>
<td>[40–60]</td>
<td>50</td>
<td>55</td>
<td>(45–55)</td>
</tr>
<tr>
<td>6</td>
<td>Almost good</td>
<td>[50–70]</td>
<td>60</td>
<td>65</td>
<td>(55–65)</td>
</tr>
<tr>
<td>7</td>
<td>Good</td>
<td>[60–80]</td>
<td>70</td>
<td>75</td>
<td>(65–75)</td>
</tr>
<tr>
<td>8</td>
<td>Very good</td>
<td>[70–90]</td>
<td>80</td>
<td>85</td>
<td>(75–85)</td>
</tr>
<tr>
<td>9</td>
<td>Excellent</td>
<td>[80–100]</td>
<td>90</td>
<td>-</td>
<td>(85–100)</td>
</tr>
</tbody>
</table>
The creation of fuzzy rules is carried out in two steps. Firstly, the algorithm automatically computes all permutations of input variables $X_j$ and their fuzzy sets $f_j$, resulting in absolutely all possible combinations. In such a case, the number of rules is $5^n$, hence having an exponential dependency on the number of input variables. The logical operator between each pair of variables is ‘and’, which then leads to the following description:

$$\begin{align*}
&f_1 \text{ AND } f_2 \text{ AND } \ldots \text{ AND } f_n \\
&f_1 \text{ AND } f_2 \text{ AND } \ldots \text{ AND } f_n \\
&\vdots \\
&f_1 \text{ AND } f_2 \text{ AND } \ldots \text{ AND } f_n \nonumber
\end{align*}$$

Subsequently, FLAP has to associate the correct output of fuzzy sets with the list described in (5). However, in order to do so the algorithm takes advantage of expert knowledge, which produces the required definitions for each input variable’s weight and bias function. The weight of each input variable $X_j$ is marked as $W_j$ and represents the degree of influence in every rule. Thus, the sum of weights is equal to the overall range of the output fuzzy sets. The bias functions interpret the expert logic by dividing each $W_j$ into sub-weights $w_{j}^{l}$ for each $f_{k}^{l}$ by using a defined coefficient $T_{k}^{l} \in [0, 1]$ (see Table 2). In regard to Table 1, let us consider $W = [0, 100]$, while $(W^j, T_k^l) \in \mathbb{R}^+$ and $\sum_{j=1}^{n} W^j = 100$. Each sub-weight $w_{j}^{l}$ is then calculated by:

$$w_{j}^{l} = W^j \cdot T_k^l, \quad (\forall w_{j}^{l} \in X^j)$$

\[ (6) \]
FLAP allows for some pre-defined choices and features multiple bias functions that express common expert logic. However, it is possible to set custom functions according to a different logic.

Table 2 Description of FLAP’s featured bias functions

<table>
<thead>
<tr>
<th>Logic</th>
<th>Bias function</th>
<th>Values of T</th>
</tr>
</thead>
<tbody>
<tr>
<td>The lower, the better (and vice versa)</td>
<td>Linear descending</td>
<td>$T_1 = 1$, $T_2 = 0.75$, $T_3 = 0.5$, $T_4 = 0.25$, $T_5 = 0$</td>
</tr>
<tr>
<td>The higher, the better (and vice versa)</td>
<td>Linear ascending</td>
<td>$T_1 = 0$, $T_2 = 0.25$, $T_3 = 0.5$, $T_4 = 0.75$, $T_5 = 1$</td>
</tr>
<tr>
<td>The more in the middle, the better (and vice versa)</td>
<td>Parabolic concave</td>
<td>$T_1 = 0$, $T_2 = 0.5$, $T_3 = 1$, $T_4 = 0.5$, $T_5 = 0$</td>
</tr>
<tr>
<td>The more in the margins, the better (and vice versa)</td>
<td>Parabolic convex</td>
<td>$T_1 = 1$, $T_2 = 0.5$, $T_3 = 0$, $T_4 = 0.5$, $T_5 = 1$</td>
</tr>
</tbody>
</table>

Once computed, the sub-weights replace the sets in list (5) according to indices $j$ and $k$.

Additionally, the logical operator ‘and’ is replaced by the operator $+$. 

$$w_1^1 + w_1^2 + \ldots + w_1^5 = \theta_1$$

$$w_1^2 + w_2^2 + \ldots + w_2^5 = \theta_2$$

$$\vdots \quad \vdots \quad \vdots$$

$$w_1^5 + w_2^5 + \ldots + w_5^5 = \theta_5,$$

(7)

The sum of sub-weights is represented by $\theta_u (u = 1, \ldots, 5)$; thus, in our case $0 \leq \theta_u \leq 100$. According to Figure 4, we consider an output variable $Y = \{y_1, \ldots, y_p\}$, $\in \mathbb{N}^+$, which consists of $l = 9$ membership functions $\mu_{h_k} (k = 1, \ldots, l)$. Hence, each interception between two membership functions are defined by the following:

$$I_k = \{ \mu_{h_k} (y_m) = 0.5 : y_m > y_m \in \{ \mu_{h_k} (y_m) = 1 \} \}$$

$$1 \leq k \leq l - 1, 1 \leq (m, \bar{m}) \leq \bar{p}$$

(8)

where $\bar{p}$ is the index of the last value in each output fuzzy set, and $y_m$ is the value where the membership function reaches 1, thus, forming the constraint, specifically for triangle functions that always have two intercepts, to consider only the right one (with a higher value). Finally, we associate the correct output fuzzy set to each rule by the following:

$$\bar{k}_u = \begin{cases} 1, & \theta_u \in [0, \bar{l}_1] \\ 2, & \theta_u \in (\bar{l}_1, \bar{l}_2] \\ \vdots & \vdots \\ 9, & \theta_u \in (\bar{l}_9, 100] \end{cases}$$

(9)

For the sake of illustration, let us take an example where four input variables have the following settings.
Table 3  Example of input variable settings

<table>
<thead>
<tr>
<th>Index/weight</th>
<th>Quality rank</th>
<th>Input 1</th>
<th>Input 2</th>
<th>Input 3</th>
<th>Input 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Very low</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>Low</td>
<td>15</td>
<td>7.5</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>Medium</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>High</td>
<td>5</td>
<td>22.5</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>Very high</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>40</td>
</tr>
</tbody>
</table>

Bias function:
- Linear descending
- Linear ascending
- Parabolic concave
- Parabolic convex

Suppose there is a combination that forms the following rule – Input 1 IS Very Low AND Input 2 IS Very Low AND Input 3 IS Medium AND Input 4 IS High. The correct output fuzzy set is then computed by using criterion (9) and Table 3. Firstly, \( \theta_u = 20 + 0 + 10 + 20 = 50 \), and then by using Table 1, it is concluded that \( 50 \in [I_4 = 45, I_5 = 55] \rightarrow k_u = 5 \). Hence, the rule is completed with – THEN Output IS Medium.

Action 2.3 utilises the fuzzy inference system that was built in action 1.5 to classify the required data. Let us consider a test dataset, which has the same \( n \) input variables \( \tilde{X}^j = \{ \tilde{x}_1, \ldots, \tilde{x}_q \}, q \in \mathbb{N}^+ (\mathbb{N}^+ = \{q > 0 : q \in \mathbb{Z}^+ \}) \) as in the training set, so that \( 2 \leq n \leq 6 \) and \( j = 1, \ldots, n \). According to Figure 1, a fuzzification procedure is initially performed, so that each datum \( \tilde{x}_i (i = 1, \ldots, q) \) in every input variable \( \tilde{X}^j \) receives a membership degree by an appropriate membership function \( \mu f^j_k (k = 1, \ldots, 5) \), as defined in the following:

\[
\tilde{x}_i = \{ \mu f^j_1 (\tilde{x}_i), \mu f^j_2 (\tilde{x}_i), \ldots, \mu f^j_5 (\tilde{x}_i) : \mu f^j_k (\tilde{x}_i) > 0 \} \tag{10}
\]

Hence, \( \tilde{x}_i \) is the fuzzy number that represents the real value \( \tilde{x}_i \) in the input variable space, when there are at most two membership functions that map it (considering the manner of their construction). After the mapping \( \tilde{x}_i \rightarrow \tilde{x}_i \) is completed for each \( \tilde{X}^j \), the system scans the rules list described in (11) for every observation \( i \) in the dataset. Thus, it locates the active rules where every input variable \( \tilde{X}^j = \{ \tilde{x}_1, \ldots, \tilde{x}_q \} \) has \( \tilde{x}_i > 0 \).

\[
\begin{align*}
& \text{IF } \tilde{X}^1 \in f^1_1 \quad \text{AND} \quad \tilde{X}^2 \in f^2_1 \quad \cdots \quad \tilde{X}^n \in f^n_1 \quad \text{THEN} \quad \tilde{Y}^1 \in h_{k_1} \\
& \text{IF } \tilde{X}^1 \in f^1_2 \quad \text{AND} \quad \tilde{X}^2 \in f^2_2 \quad \cdots \quad \tilde{X}^n \in f^n_2 \quad \text{THEN} \quad \tilde{Y}^2 \in h_{k_2} \\
& \quad \vdots \quad \quad \quad \quad \quad \vdots \quad \quad \quad \quad \quad \vdots \quad \quad \quad \quad \quad \vdots \\
& \text{IF } \tilde{X}^1 \in f^1_5 \quad \text{AND} \quad \tilde{X}^2 \in f^2_5 \quad \cdots \quad \tilde{X}^n \in f^n_5 \quad \text{THEN} \quad \tilde{Y}^5 \in h_{k_5} \\
\end{align*}
\tag{11}
\]

Consequently, the fuzzy output is composed of multiple membership functions \( S(\tilde{Y}) = \{ \mu h_{k_1} (\tilde{Y}^1), \ldots, \mu h_{k_5} (\tilde{Y}^5) \} \), where \( 1 \leq \omega \leq 5 \). These membership functions have an upper membership degree limit, since \( \tilde{Y}^\omega \in (0, 1] \) is obtained via the inference criterion
Proposal and examination of the FLAP algorithm

\( \tilde{y}^u = \min_u (\tilde{X}^1, \ldots, \tilde{X}^u) \) applied on each active rule indexed by \( u \). Hence, the final fuzzy output is basically the envelope or union of areas under the capped membership functions:

\[
S\{\tilde{y}\} = S\left( \bigcup_u \mu_h \{\tilde{y}^u\}, \forall \tilde{u} \right) = \bigcup_u \left( \left\{ \int \mu_h \{\tilde{y}^u\} d\gamma, \forall \tilde{u} \right\} \right)
\]  
(12)

Finally, after computing \( S\{\tilde{y}\} \), the algorithm uses a defuzzification procedure (Ross, 2004) to achieve the crisp result \( y_i \), where in our case \( 0 \leq y_i \leq 100 : y_i \in \mathbb{R}^+ \).

3.3 Description of FLAP-SLM (self-learning module)

FLAP-SLM is a very important part of the algorithm, and in fact, constitutes its initial phase. Basically, the module can be considered a semi-supervised meta learning algorithm that compares the results of various FLAP configurations with a relatively small classified sample. Its sole objective is to find a specific configuration of methods that will make FLAP classify a required dataset with minimal error. FLAP-SLM has a very similar flow to that in Figure 2, but obviously, there is a slight difference. After the pre-processing action (1.2 in Figure 2), an additional action is performed. This action splits the training set into two new datasets, one for training and the other for testing, so that the newly sampled test set will be as similar as possible to the original one. More precisely, the module searches for a sample that satisfies the minimal Mahalanobis distance

\[
d(O_i, O_i') = \sqrt{(O_i - O_i')^TC^{-1}(O_i - O_i')}
\]

while also conducting a two-sided Kolmogorov-Smirnov test and keeping a minimal statistic

\[
D_{\tilde{x}_i, \tilde{x}_i'} = \sup \left| F_{\tilde{x}_i}(x) - F_{\tilde{x}_i'}(x) \right| \text{ with a high p-value.}
\]

When \( O_i \) and \( O_i' \) are the observations and \( C \) is the covariance matrix, \( F_{\tilde{x}_i}(x) \) and \( F_{\tilde{x}_i'}(x) \) are the cumulative distribution functions of the original and sampled test sets, respectively. It is also important to state that this sampled test set should already be classified by either an expert or a credible expert system. FLAP’s core actions (1.3, 1.4, 1.5 and 2.3 in Figure 2) then follow an iterative process over the newly sampled datasets: each time a new classifier is built and executed according to a different configuration of algorithm options.

As noted in Table 4, there are quite a few choices for each core action component. Accordingly, the maximum total amount of configurations is

\[
cq_{\text{total}} = cq_{W}/ \cdot \left( cq_{\text{KMPF}} + cq_{\text{PAM}} + cq_{\text{FCM}} + cq_{\text{HAC}} \right) \cdot cq_{D} \cdot cq_{\text{MAD}},
\]

and since all variables besides \( cq_{W}/ \) are constant, it can be rewritten as \( cq_{\text{total}} = cq_{W}/ \cdot 660 \). Therefore, the amount of configurations, and hence also the module’s runtime, mainly depend on the number of input variables. In order to illustrate the magnitude of influence on algorithm runtime, three experiments were conducted on a demo dataset.
Table 4  List of algorithm components and their tested FLAP-SLM options

<table>
<thead>
<tr>
<th>Algorithm component</th>
<th>Tested options</th>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAD method</td>
<td>$\alpha = {2, 2.5, 3} \text{ or disabled}$</td>
<td>$c\Omega_{MAD} = 4$</td>
</tr>
<tr>
<td>KMPP model</td>
<td>-</td>
<td>$c\Omega_{KMPP} = 1$</td>
</tr>
<tr>
<td>PAM model</td>
<td>-</td>
<td>$c\Omega_{PAM} = 1$</td>
</tr>
<tr>
<td>FCM model</td>
<td>$m = {1.5, 2, 2.5}$</td>
<td>$c\Omega_{FCM} = 3$</td>
</tr>
<tr>
<td>HAC model</td>
<td>$Distance = {\text{Euclidean, Manhattan, Maximum, Canberra}}$</td>
<td>$c\Omega_{HAC} = 28$</td>
</tr>
<tr>
<td></td>
<td>$Merging\ method = {\text{single link, complete link, average link, McQuitty, median, centroid, ward}}$</td>
<td></td>
</tr>
<tr>
<td>Defuzzification method</td>
<td>$D = {\text{centroid, bisector, SOM, LOM, MOM}}$</td>
<td>$c\Omega_{D} = 5$</td>
</tr>
<tr>
<td>Input variable weights</td>
<td>$W^j = {10, \ldots, (110 - 10 \cdot n)}$, $\mod\left(\frac{W^j}{10}\right) = 0$, $2 \leq n \leq 6$</td>
<td>$c\Omega_{W^j} = \begin{cases} 9, &amp; n = 2 \ 36, &amp; n = 3 \ 84, &amp; n = 4 \ 126, &amp; n = 5 \ 126, &amp; n = 6 \end{cases}$</td>
</tr>
</tbody>
</table>

Figure 5  Amount of configurations and runtime (in hours) in accordance with the number of input variables. RTexp_1: an experiment with a training set of 3,004 observations and a test set of 306 observations. RTexp_2: an experiment with a training set of 707 observations and a test set of 306 observations. RTexp_3: an experiment with a training set of 3,004 observations and a test set of 72 observations.

As illustrated in Figure 5, the FLAP-SLM runtime increases significantly after three input variables; after six, it already reaches several hundred hours. Therefore, the algorithm is limited to six input variables. The size of the training set also has a fairly substantial influence on the runtime, although it is slightly weaker than that of the test set.

In the final stage, FLAP-SLM compares the classifiers’ results with the classified sample, by using different precision indices, depending on whether the output variable is ordinal or numeric.
Proposal and examination of the FLAP algorithm

Let us consider an output variable \( Y_{cq} = \{y_1, \ldots, y_p\} \) for each classifier that is constructed from a configuration \( cq = 1, \ldots, cq_{total} \) when \( cq_{total} \) is computed via Table 4. Additionally, let there be a test sample that was taken from the original training set, in which \( V = \{v_1, \ldots, v_r\} \) is the vector of the expert classifications; and \( v_i \) is the classification of observation \( i \) that represents either a continuous numeric grade (in our case in \([0, 100]\) range) or an ordinal class with multiple categories. If the output variable is ordinal, the module uses a kappa (Cohen, 1960) or weighted kappa (Cohen, 1968) statistic for comparison. Hence, a cross-classification table is computed for each pair of \( Y_{cq} \) and \( V \), where \( \overline{p} \) represents the ratio between \( Y_{cq} \) and \( V \) for every pair of categories:

<table>
<thead>
<tr>
<th>( Y_{cq} )</th>
<th>Category 1</th>
<th>Category 2</th>
<th>...</th>
<th>Category ( \varepsilon )</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V )</td>
<td>( \overline{p}_1 )</td>
<td>( \overline{p}_2 )</td>
<td>...</td>
<td>( \overline{p}_\varepsilon )</td>
<td>( \overline{p}_v )</td>
</tr>
<tr>
<td>Category 1</td>
<td>( \overline{p}_{11} )</td>
<td>( \overline{p}_{12} )</td>
<td>...</td>
<td>( \overline{p}_{1\varepsilon} )</td>
<td>( \overline{p}_1 )</td>
</tr>
<tr>
<td>Category 2</td>
<td>( \overline{p}_{21} )</td>
<td>( \overline{p}_{22} )</td>
<td>...</td>
<td>( \overline{p}_{2\varepsilon} )</td>
<td>( \overline{p}_2 )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Category ( \varepsilon )</td>
<td>( \overline{p}_{\varepsilon 1} )</td>
<td>( \overline{p}_{\varepsilon 2} )</td>
<td>...</td>
<td>( \overline{p}_{\varepsilon \varepsilon} )</td>
<td>( \overline{p}_\varepsilon )</td>
</tr>
<tr>
<td>Total</td>
<td>( \overline{p}_1 )</td>
<td>( \overline{p}_2 )</td>
<td>...</td>
<td>( \overline{p}_\varepsilon )</td>
<td>1</td>
</tr>
</tbody>
</table>

According to Table 5, if \( \varepsilon = 2 \), then the kappa is computed. Otherwise, the weighted kappa is computed with weights equal to the squared distances between the categories. Configurations are then sorted by the calculated statistic, but only ten configurations with the highest values are eventually selected. In the case of numeric output, the module computes the \( RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \tilde{y}_i)^2}{n}} \) statistic for every \( Y_{cq} \). Similarly to the previous process, all configurations are then sorted, but only ten configurations with the lowest values are selected. Furthermore, in order to test the values’ stability, confidence intervals are computed for both numeric and ordinal indices. This is done via an advanced bootstrapping method called bias-corrected and accelerated (BCa) (DiCiccio and Efron, 1996; Myers and Hsueh, 2001), when the number of random samples \( \beta = 1,000 \) (can be changed in the algorithm) and confidence level \( \alpha_{BCa} = 0.05 \).

Finally, we hypothesise that the most accurate FLAP configuration is among the top ten configurations, and since the algorithm is deterministic, it would be able to classify the original test dataset with similar precision. In many cases, it would be possible to notice a certain tendency in the top ten configurations because most of them are likely to be very similar. FLAP-SLM utilises the information obtained from these configurations, while it concentrates on the input variables’ weights. Firstly, in order to find the most dominant variable it calculates the sum of weights for each one. Subsequently, it picks the highest weight in that chosen variable, out of the given ten, which ultimately indicates the supposedly most accurate configuration. At this stage, the module has finished its learning procedure; hence, FLAP is activated with the chosen configuration on the original training and test datasets.
3.4 Experimental study

In order to examine the performance of FLAP in a practical classification problem, a comparative test was conducted on a dataset in the medical field (Pima Indians Diabetes Data Set, 1990). This particular dataset was deliberately selected due to an accessible expert. The dataset contains eight input variables that describe various medical tests and physical parameters by which 768 women of Pima Indian heritage were classified for the presence of diabetes. Specifically for the sake of the test, all observations were already classified with a categorical variable stating either ‘positive’ or ‘negative’. It is important to note that some input variables contained irrational data indicated by zeroes, for example zero diastolic blood pressure or zero body mass index. Thus, it was decided to ignore any observation that had at least one illogical value in any of the relevant variables. This decision decreased the number of observations to 392.

Figure 6   Probability density functions of input variables (see online version for colours)

Since FLAP can work with no more than six input variables that also require bias functions, a medical professor from Assaf HaRofeh Medical Center was consulted and asked to provide an expert opinion. Our specialist selected six input variables that are basically enough for diabetes diagnostics. Additionally, he defined the following logic.

The output variable consists of 33.16% ‘positive’ and 66.84% ‘negative’ observations. Similarly, as shown in Table 1 and Figure 4, the output variable was defined in FLAP by nine fuzzy sets in a [0, 100] range. The categories were defined as – ‘negative’: [0, 50] and ‘positive’: [50, 100]. Since the maximum amount of input variables had been selected, it was decided to decrease the number of possible configurations to achieve a reasonable runtime. This minimisation was done through trial and error, while testing the algorithm multiple times with different options and comparing classifications’ precision. Eventually, a narrow set of options was chosen.
Proposal and examination of the FLAP algorithm

Table 6  Definition of input variables

<table>
<thead>
<tr>
<th>Input variable</th>
<th>Expert logic</th>
<th>Appropriate bias function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of times pregnant</td>
<td>The higher the value, the less chance for disease</td>
<td>Linear descending</td>
</tr>
<tr>
<td>Plasma glucose concentration in an oral</td>
<td>The higher the value, the more chance for disease</td>
<td>Linear ascending</td>
</tr>
<tr>
<td>glucose tolerance test (a two hours)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diastolic blood pressure (mm Hg)</td>
<td>The higher the value, the more chance for disease</td>
<td>Linear ascending</td>
</tr>
<tr>
<td>Triceps skin fold thickness (mm)</td>
<td>The higher the value, the more chance for disease</td>
<td>Linear ascending</td>
</tr>
<tr>
<td>Two-hour serum insulin (mu U/ml)</td>
<td>The higher the value, the less chance for disease</td>
<td>Linear descending</td>
</tr>
<tr>
<td>Body mass index [weight in kg/(height in m)^2]</td>
<td>The higher the value, the more chance for disease</td>
<td>Linear ascending</td>
</tr>
</tbody>
</table>

Table 7  List of selected algorithm components and their options

<table>
<thead>
<tr>
<th>Algorithm component</th>
<th>Tested options</th>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAD method</td>
<td>α = 2.5</td>
<td>cq_{MAD} = 1</td>
</tr>
<tr>
<td>HAC model</td>
<td>Distance = Euclidean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Merging method = {complete link, median}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cq_{HAC} = 2</td>
<td></td>
</tr>
<tr>
<td>Defuzzification method</td>
<td>D = {bisector, MOM}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cq_{D} = 2</td>
<td></td>
</tr>
<tr>
<td>Input variable weights</td>
<td>W'/ = {10, ..., (110 - 10 \cdot n)}, \mod\left(\frac{W'}{10}\right) = 0, 2 \leq n \leq 6</td>
<td>cq_{W'} = \begin{cases} 9, &amp; n = 2 \ 36, &amp; n = 3 \ 84, &amp; n = 4 \ 126, &amp; n = 5 \ 126, &amp; n = 6 \end{cases}</td>
</tr>
</tbody>
</table>

It was decided to let FLAP compete with six different and well-known supervised learning algorithms – ANN, random forest, SVM, Adaboost.M1, bagging, and weighted k-nearest neighbour (kNN). Thus FLAP, as a semi-supervised learning algorithm, was compared in the best possible way. While all other algorithms trained on the entire classified dataset, FLAP only trained on a sample from it.

The test itself was performed using a ten-fold cross-validation technique. In this common validation technique, the dataset is randomly divided into ten equal-sized partitions; thus, the algorithms train on nine partitions and then evaluate the tenth. This procedure is repeated ten times and each of the different partitions is evaluated only once. The performance of every algorithm was assessed in each iteration by the kappa statistic. Subsequently, in order to prove dissimilarity among the algorithms, a two-tailed Friedman test was performed with a confidence level of \( p = 0.05 \). Furthermore, after computing the Friedman statistic, the Iman-Davenport modification was also added. The Friedman test is basically a non-parametric and rank-based test that has two hypotheses.
**4. Results**

At first, due to the relatively small dataset, we could not achieve a significant result in the Friedman test. In other words, we were not able to reject the null hypothesis. Therefore, instead of simply increasing the number of folds and thus decreasing the size of each fold, it was decided to repeat the ten-fold cross-validation several times. This method is valid, owing to the absolutely random allocation of data to each fold. Eventually, after six iterations, we successfully rejected the null hypothesis. Tables 8 and 9 show the final results of our examination.

### Table 8 Average ranks of the algorithms according to each tested scenario

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>All with six input variables</th>
<th>Only FLAP with three input variables</th>
<th>All with three input variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average rank</td>
<td>Algorithm</td>
<td>Average rank</td>
</tr>
<tr>
<td>Random forest</td>
<td>2.9545</td>
<td>FLAP</td>
<td>3.0000</td>
</tr>
<tr>
<td>Bagging</td>
<td>3.5378</td>
<td>Random forest</td>
<td>3.1048</td>
</tr>
<tr>
<td>SVM</td>
<td>3.8863</td>
<td>Bagging</td>
<td>3.3387</td>
</tr>
<tr>
<td>kNN</td>
<td>4.2121</td>
<td>kNN</td>
<td>3.9838</td>
</tr>
<tr>
<td>FLAP</td>
<td>4.2575</td>
<td>SVM</td>
<td>4.5564</td>
</tr>
<tr>
<td>ANN</td>
<td>4.8863</td>
<td>ANN</td>
<td>5.2258</td>
</tr>
</tbody>
</table>
Table 9 Critical values of the tests according to each tested scenario

<table>
<thead>
<tr>
<th></th>
<th>All with six input variables</th>
<th>Only FLAP with three input variables</th>
<th>All with three input variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friedman statistic (B value)</td>
<td>5.2469</td>
<td>11.8275</td>
<td>5.5047</td>
</tr>
<tr>
<td>F critical value</td>
<td>2.444</td>
<td>2.444</td>
<td>2.444</td>
</tr>
<tr>
<td>Nemenyi critical value</td>
<td>1.1631</td>
<td>1.1631</td>
<td>1.1631</td>
</tr>
</tbody>
</table>

It is possible to notice in Table 9 that $B > F_{critical}$ in all scenarios. Therefore, the null hypothesis was rejected. Consequently, the critical difference value was computed and then used as a threshold in order to distinguish the pairs of algorithms that have a statistically significant dissimilarity. Finally, the matrix of average ranks’ differences was created, as shown in Figures 7, 8 and 9.

Figure 7 Matrix of average ranks’ differences (all with six input variables) (see online version for colours)
Figure 8  Matrix of average ranks’ differences (all with three input variables) (see online version for colours)

![Coefficient Matrix for All Methods](image1.png)

Figure 9  Matrix of average ranks’ differences (only flap with three input variables) (see online version for colours)

![Coefficient Matrix for Flap Only](image2.png)

Additionally, Table 10 shows the overall FLAP performance.
Proposal and examination of the FLAP algorithm

Table 10  FLAP performance indices according to each tested scenario

<table>
<thead>
<tr>
<th></th>
<th>All with six input variables</th>
<th>Only FLAP with three input variables</th>
<th>All with three input variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>0.7355</td>
<td>0.7606</td>
<td>0.7517</td>
</tr>
<tr>
<td>Kappa statistic</td>
<td>0.4147</td>
<td>0.4684</td>
<td>0.4557</td>
</tr>
<tr>
<td>False-positive ratio</td>
<td>0.2169</td>
<td>0.1946</td>
<td>0.2143</td>
</tr>
<tr>
<td>False-negative ratio</td>
<td>0.3602</td>
<td>0.3294</td>
<td>0.3166</td>
</tr>
</tbody>
</table>

5 Discussion

Considering the results in the previous section, we can confidently say that FLAP is fairly accurate when compared to the other algorithms. Judging by the average ranks presented in Table 8, it is definitely not a surprise that bagging, SVM, and particularly random forest are the leading algorithms in the experiment. However, while random forest with six input variables was proven to be significantly superior to the others, it dropped by a few ranks when tested with less input variables. Even though it was not proven significant, FLAP exceeded these algorithms and successfully climbed to the top position with merely half of the input variables. Interestingly enough, with just three input variables, FLAP was able to achieve an even higher agreement rate, while also demonstrating a significantly better performance than ANN, kNN, AdaBoost.M1 and even SVM. Despite not having the privilege to train on the whole classified set, FLAP still managed to produce some significant results and basically reach the same standard set by the algorithms. Hence, unlike the others, there is a strong possibility that FLAP could be more applicable to practical classification problems that have limited classified data. Additionally, it is important to mention that with less input variables, FLAP has considerably less configurations (approximately 71.4% less with half of the variables), thus also a substantially shorter runtime. Notwithstanding that a thorough inspection was made by trial and error, there could still be a chance that some unselected algorithm options or components would make FLAP achieve an even better performance level. Yet, in order to substantiate this claim, more tests should be made. Thus, researchers should also examine whether FLAP is able to produce even more satisfying results in other fields. Furthermore, some other notably interesting outcomes are the false-positive and false-negative ratios. It was clearly shown in Table 10 that the algorithm tends to a lower false-positive and a higher false-negative. Unfortunately, this condition is definitely not suitable for the medical field, since a higher false-negative is considered more severe than false-positive. However, in many other fields it can actually be the opposite. A good example can be virus protection software that sends false warnings, often causing users to quickly lose faith in the program and ultimately withdraw. Another example is the banking field, where false classification of credit fraud may cause an inexcusable credit hold.
6 Conclusions

We may conclude that none of the tested algorithms in this experiment is qualified for usage in this specific medical classification problem. The main reason for this is the relatively low agreement rate and the high false-negative ratio. Nevertheless, from a comparative perspective, we can definitely see that FLAP has significant classification potential, specifically due to its inference methodology. Moreover, its low dependency on classified data is extremely attractive. However, in order to achieve a comprehensive understanding of the algorithm abilities, more tests are needed. Additionally, considering our study of the algorithm, we assume that a few essential modifications can substantially improve the current lack in scalability and efficiency. Hence, we offer two main courses of improvement. Firstly, in order to significantly decrease the runtime, it is advised to integrate an optimisation algorithm, such as PSO or GA into the learning module. Secondly, it is vital to combine an automatic method to minimise the number of fuzzy rules, thus removing the input variable limitation.

References

Proposal and examination of the FLAP algorithm


Notes

1 Although the number of input variables is theoretically unlimited, the algorithm is capped at six, due to a practical reason described in Section 3.3.
2 The algorithm may skip a configuration when at least one method or model cannot be computed, due to the nature of the data.

3 Plasma glucose concentration a two hours in an oral glucose tolerance test, diastolic blood pressure, triceps skin fold thickness, two-hour serum insulin, body mass index.