UniFeat

Universal Feature Selection Tool

User Manual, Version 1.0

Sina Tabakhi, Parham Moradi

June 2022

Contents

Acknowledgments

The UniFeat is implemented and developed principally by Sina Tabakhi and Parham Moradi, but other people make essential contributions. We are grateful to Mr. Shahin Salavati, who implemented most of the graphical user interfaces. We also thank the professors who provided us with feedback about the tool, especially Dr. Fardin Akhlaghian at the University of Kurdistan and Dr. Haiping Lu at the University of Sheffield.

About the Authors

Sina Tabakhi is currently a Ph.D. student in the Department of Computer Science at the University of Sheffield, United Kingdom, working in the Machine Learning Research Group. Before joining the University of Sheffield, he had worked in the industry as a software engineer for five years. He graduated with a Master's degree as the first honor student in Computer Engineering, Artificial Intelligence from the University of Kurdistan, Iran. He has been working on efficient feature selection methods for integrative multi-omics analysis to tackle the curse of the dimensionality phenomenon of these data types.

Parham Moradi received the M.Sc. and Ph.D. degrees in Computer Science from Amirkabir University of Technology, Iran, in 2005 and 2011, respectively. He conducted a part of his Ph.D. research work in the Laboratory of Nonlinear Systems, Ecole Polytechnique Federal de Lausanne (EPFL), Lausanne, Switzerland, from September 2009 to March 2010. He is currently an associate professor in the Department of Computer Engineering at the University of Kurdistan, Iran. His research focuses on machine learning, social network analysis, recommender systems, and deep learning.

1 Introduction

The field of data mining is concerned with knowledge discovery from data through the development of computer programs. During the last two decades, the rapid advances in data acquisition capacity and database technology have led to the production of datasets with large numbers of features in many fields [9]. Most of the features are irrelevant and redundant, and these unnecessary features have stimulated a phenomenon in the data mining methods [6]. Therefore, data preprocessing plays an essential role in the data mining domain. Feature selection is regarded as an important and active research area in data preprocessing.

Feature selection is the process of identifying a subset of relevant features in an original feature set [27, 26]. Up to now, many feature selection methods have been proposed. From one aspect, these methods can be classified into four approaches, including filter, wrapper, embedded, and hybrid [29, 21, 22]. The filter approach estimates the relevance of features using intrinsic properties of the data without the need for any learning algorithms. This approach can be subdivided into univariate and multivariate [29, 25, 2]. The univariate filter approach examines the relevance of each feature individually based on a given criterion. In contrast, the multivariate filter approach selects a subset of features by considering the dependencies between features. The wrapper approach integrates a specific learning algorithm to evaluate different subsets of selected features within the feature selection process. The embedded approach considers the feature selection process as part of constructing a given learning algorithm. Finally, the hybrid approach uses filter-based methods to reduce the original feature set in the first step and then applies wrapper-based techniques to select the final feature set.

From another aspect, the feature selection methods can be classified into supervised and unsupervised modes [9, 21]. In the supervised mode, the class labels of data are applied in the feature selection process as a guide, but in the unsupervised mode, the process of feature selection is done without using class labels.

Although many feature selection methods based on different approaches have been proposed in the literature, no attempt has been made to develop a tool bringing well-known and state-of-the-art methods together as a benchmark for a comprehensive comparison of their performance. Besides, most current tools focus on only one approach and ignore the others. Furthermore, some researchers prefer to use graphical user interfaces (GUIs) rather than command-line environments, but most of the available tools do not provide any GUIs (See Section 2 for more details about the shortcomings of the previous tools).

Thus, our aim to develop the Universal Feature Selection Tool (UniFeat) as a comprehensive feature selection tool includes six aspects. (1) UniFeat implements well-known and state-of-the-art feature selection methods within a unified framework. (2) UniFeat can be considered a benchmark tool due to the development of methods in all the approaches. (3) The functions presented in UniFeat provide essential auxiliary tools needed for performance evaluation, results visualization, and statistical analysis. (4) UniFeat has been entirely implemented in Java and can be run on various platforms. (5)

Researchers are able to use UniFeat through its GUI environment or as a library in their Java codes. (6) Finally, the open-source nature of UniFeat can help researchers use and modify the tool to fit their research requirements and greatly facilitate them to share their methods with the scientific community rapidly.

2 Background

Several tools have been developed so far for the feature selection task. We compared UniFeat to existing tools that have implemented feature selection methods, and the comparison results are provided in Table 2.1. Based on the main focuses of these tools, they can be classified into two categories: data mining and feature selection tools. Data mining tools provide a general-purpose environment for machine learning models, and feature selection can be considered only a small part of these tools. Weka [12], RapidMiner [14], Mulan [37], MLC++ [18], and Spider[1] are well-known data mining tools. Weka is general-purpose software for data mining, but its feature selection part does not consist of representative and state-of-the-art methods. It provides only a few conventional feature selection methods based on the filter and wrapper approaches. RapidMiner is an integrated platform for the generation of machine learning models. Many representative filter-based feature selection methods are implemented in this software, but state-of-the-art ones are missing. Furthermore, only a few baseline methods based on the wrapper and embedded approaches are available in the RapidMiner repository. Mulan is an open-source Java library that provides simple baseline feature selection methods without any GUI. MLC++ is another data mining tool developed based on C++ at Stanford University. This tool implements only some wrapper-based methods, and the other two categories have been ignored in the library. Spider is intended to be an environment for machine learning in MATLAB, while its feature selection part involves a few basic filter and embedded methods. This tool does not have any GUI and works only in the command-line environment.

On the other hand, feature selection tools are designed specifically for the feature selection task and provide various feature selection methods. The ASU feature selection repository (ASUFS)[2], GALGO [36], FEAST [4], and LOFS [40] are some available feature selection tools. ASUFS is a software package based on Python that implements many conventional and well-known filter-based feature selection algorithms. This tool also implements only traditional wrapper-based methods and neglects the embedded approach. GALGO was developed based on the R language, which focuses only on genetic algorithms as a wrapper approach. It is not an open-source tool and does not provide any GUI. FEAST is a feature selection tool developed on the basis of MATLAB and implements only standard mutual information based on filter methods without any GUI. LOFS has been released recently and is an open-source library for online streaming feature selection tasks. This tool, developed in MATLAB, consists of only a few online filter-based methods.

[1] http://people.kyb.tuebingen.mpg.de/spider/
[2] https://jundongl.github.io/scikit-feature/

Table 2.1: Comparison of UniFeat to existing feature selection tools.

Tool	Main Focus	Language	Open-source	GUI	Feature Selection Approach		
					Filter	**Wrapper**	**Embedded**
Weka [12]	Data Mining	Java	✓	✓	Consists of a small number of conventional methods.	Few baseline methods have been implemented.	-
RapidMiner [14]	Data Mining	Java	✓	✓	Many representative methods have been implemented but state-of-the-art ones are missing.	Few baseline methods have been implemented.	Few baseline methods have been implemented.
Mulan [37]	Multi-label learning	Java	✓	-	Few baseline methods have been implemented.	-	-
Spider	Data Mining	MATLAB	✓	-	Only small number of basic methods have been implemented.	-	Few baseline methods have been implemented.
MLC++ [18]	Data Mining	C++	-	-	-	Includes many representative methods.	-
GALGO [36]	Feature selection	R	-	-	-	Only genetic algorithm has been implemented.	-
ASUFS	Feature selection	Python	✓	✓	Many conventional and well-known methods have been implemented but state-of-the-art ones are missing.	Few baseline methods are provided.	-
FEAST [4]	Feature selection	MATLAB	✓	-	Provides implementations of standard mutual information-based methods.	-	-
LOFS [40]	Online feature selection	MATLAB	✓	-	-	Some online methods have been implemented.	-
UniFeat	**Feature selection**	**Java**	✓	✓	Many well-known and state-of-the-art methods have been implemented.	Consists of some well-known and state-of-the-art methods.	Some baseline and well-known methods are provided.

3 Introduction to UniFeat

The **Uni**versal **Feat**ure Selection Tool (UniFeat) is an open-source Java tool for feature selection, developed at the University of Kurdistan, Iran, and distributed under the MIT License[3] terms. The project aims to create a unified framework for researchers applying feature selection.

For simplification of the development of the tool, UniFeat was divided into six main packages, including (1) featureSelection, (2) dataset, (3) classifier, (4) gui, (5) result, and (6) util (as shown in Figure 3.1), used for the following purposes.

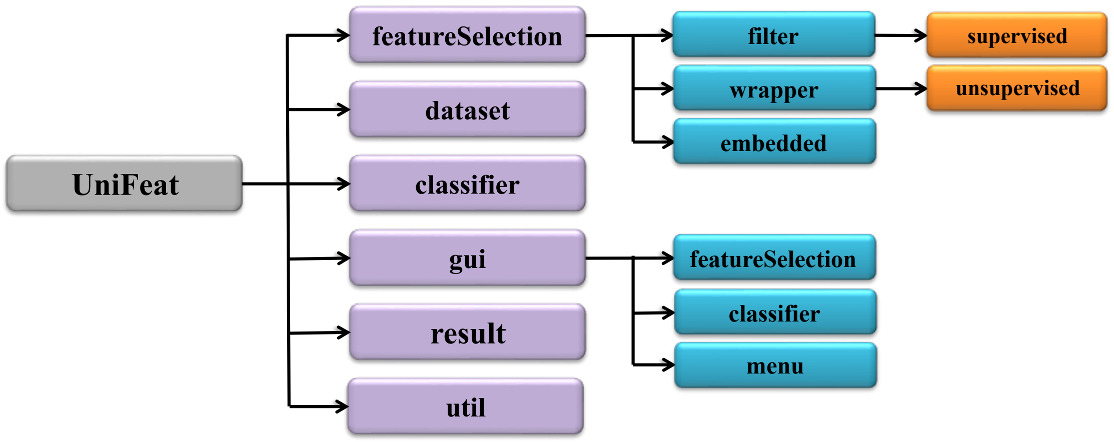


Figure 3.1: UniFeat packages.

1. **featureSelection package**: provides all the feature selection methods implemented in the tool. This package is divided into three sub-packages to cover all the feature selection approaches. Moreover, filter-based feature selection methods have been split into supervised and unsupervised packages. The current feature selection methods in the UniFeat repository are based on the filter, wrapper, and embedded approaches. The unified interface of the package allows researchers to implement their feature selection methods and share them with the other researchers in the feature selection community.
2. **dataset package**: is used for loading, saving, editing, and exporting different types of dataset files.
3. **classifier package**: collects several well-known and frequently used classifiers from the Weka software package [12].
4. **gui package**: provides GUIs that display the entire graphical representation of the panels for interaction with the user. Moreover, the package reports the results visually. It should be noted that this package has been separated from the others.
5. **result package**: reports the performance results of feature selection methods based on several criteria such as accuracy and execution time.

[3]https://opensource.org/licenses/MIT

6. **util package**: presents various utility methods for manipulating arrays and matrices and performing basic statistical operations that can be used in the feature selection methods.

UniFeat is entirely implemented in Java, and it can thus be run on any platform where Java Runtime Environment (JRE) is installed.

4 Download and run

Two types of files are provided for the UniFeat:

1. The executable file of UniFeat (version 0.1.0) that can be downloaded from the project website[4].
2. The source-code of UniFeat (version 0.1.0) which is available in the GitHub repository[5].

After downloading the tool, you must have the Java Runtime Environment (JRE) on your system to run it. Also, if you want to use the source codes and modify them, you need the Java Development Kit (JDK) to compile the modified source codes again.

You can start the UniFeat as a graphical user interface by clicking the UniFeat-v0.1.0.jar file or by typing the following command from the command prompt:

```
java -jar UniFeat-v0.1.0.jar
```

Figure 4.1 shows the initial panel of the UniFeat. This panel is used to select a workspace path for the tool. It should be noted that some essential files will be created in this path.

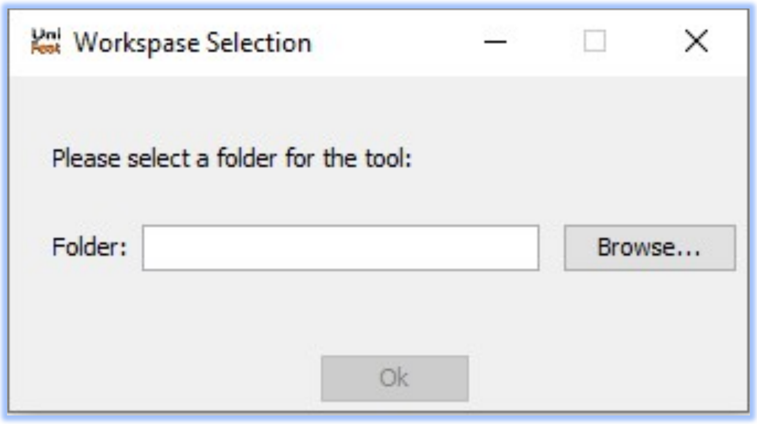


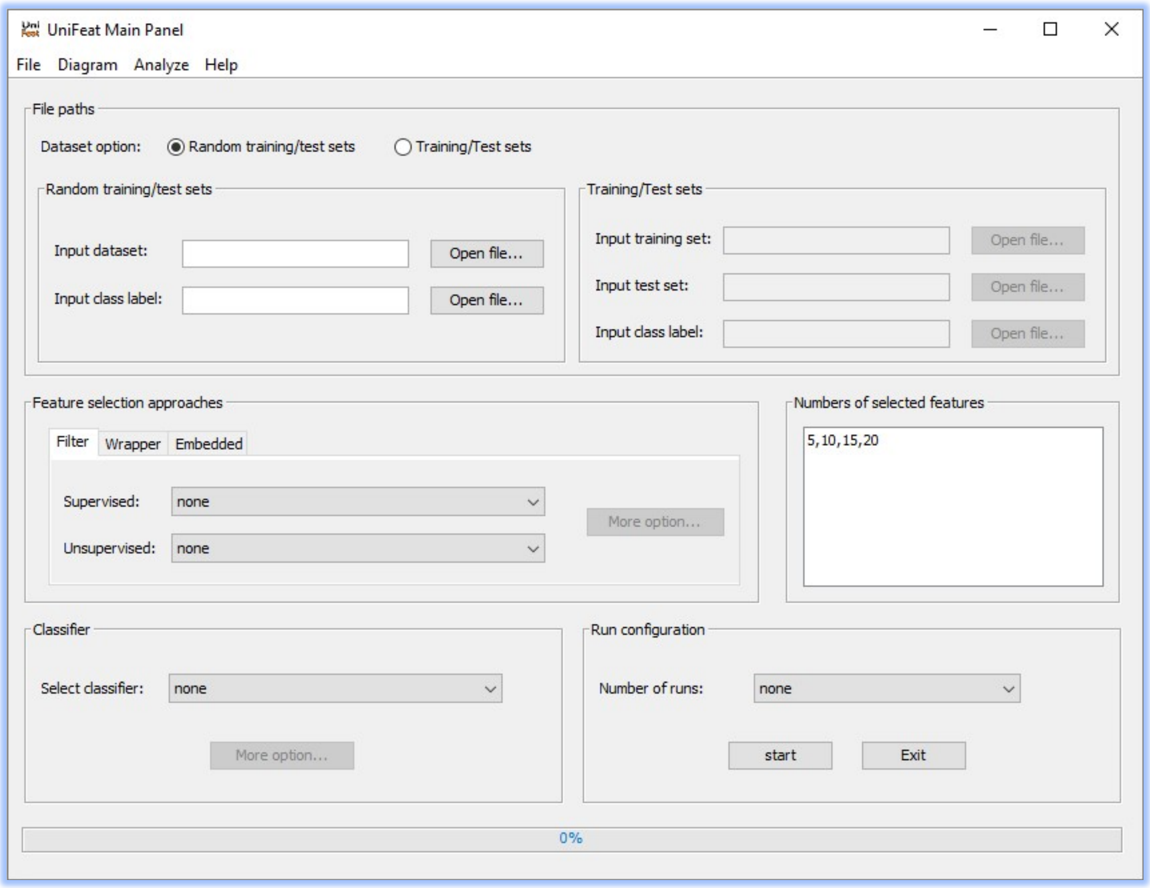
Figure 4.1: Workspace selection panel.

The easiest way to use UniFeat is through its graphical user interface. Another way is to use UniFeat as a library (using UniFeat-v0.1.0.jar) for researchers who use feature selection as a part of their own methods and, therefore, prefer to embed the UniFeat feature selection methods in their Java codes. The detailed information is described in Section 6.

[4] https://unifeat.github.io/software.html
[5] https://github.com/UniFeat/unifeat

5 The UniFeat exploration

After running UniFeat and selecting a workspace for the tool, you will see the main panel of the tool, which is illustrated in Figure 5.1. This panel gives all the tool facilities access using form filling and menu items.



UniFeat Main Panel

File Diagram Analyze Help

File paths

Dataset option: [✓ Random training/test sets] Training/Test sets

Random training/test sets

Input dataset: Open file...

Input class label: Open file...

Training/Test sets

Input training set: Open file...

Input test set: Open file...

Input class label: Open file...

Feature selection approaches

Filter Wrapper Embedded

Supervised: none

Unsupervised: none

More option...

Numbers of selected features

5,10,15,20

Classifier

Select classifier: none

More option...

Run configuration

Number of runs: none

start Exit

0%

Figure 5.1: Main panel of UniFeat.

5.1 Panel description

There are five different parts corresponding to the specific task of the UniFeat tool. Each of the five parts is described in the following subsections.

5.1.1 Loading the dataset files

In the feature selection methods, datasets are split into training and test sets. The training set is a portion of the data that is used in the feature selection process. Moreover, this set is employed for training the learning algorithm. On the other hand, the test set is the unseen data that is used to evaluate the performance of the feature selection methods. Generally speaking, benchmark datasets which are provided for the feature selection domain are available in two types. In the first type, the dataset file consists of all the samples. In the second type, the datasets are divided originally into two portions: training and test sets. UniFeat provides support to both types of dataset files.

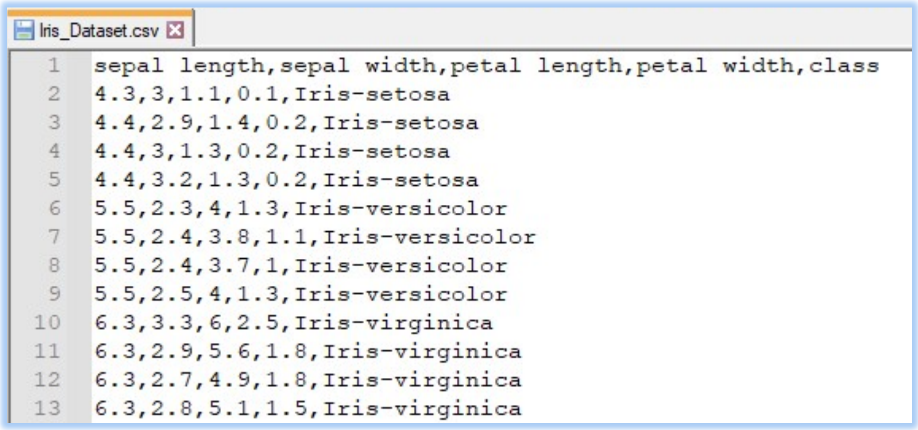
The "File paths" panel allows you to load all dataset files in the tool for future purposes:

1. "Random training/test sets" section: you can easily import a file of the dataset, and then the training and test sets are drawn randomly by the tool from the input dataset file (2/3 of the data are considered as a training set, and the other portion is used as a test set).
2. "Training/Test sets" section: if you want to use the datasets that are divided originally into training and test sets, this section can be used.

A specific way of representing datasets is needed for the tool. In UniFeat format, the representation of the input dataset consists of the following parts:

- The first row of the datasets must have the names of all features.
- The next rows contain all data, with each row corresponding to a sample. A vector of feature values describes each sample with a class label separated by commas. Also, the class labels of all samples must be available in the last column of the dataset.

You can easily import the dataset into the UniFeat tool as a file in comma-delimited format (i.e., CSV file format). Figure 5.2 shows an example of a dataset in the UniFeat format. In Figure 5.2, the input dataset contains 12 samples and four features within the class feature. In this case, the class feature has three values, including *Iris-setosa*, *Iris-versicolor*, and *Iris-virginica*.



```
sepal length,sepal width,petal length,petal width,class
4.3,3,1.1,0.1,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.4,3,1.3,0.2,Iris-setosa
4.4,3.2,1.3,0.2,Iris-setosa
5.5,2.3,4,1.3,Iris-versicolor
5.5,2.4,3.8,1.1,Iris-versicolor
5.5,2.4,3.7,1,Iris-versicolor
5.5,2.5,4,1.3,Iris-versicolor
6.3,3.3,6,2.5,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica
6.3,2.7,4.9,1.8,Iris-virginica
6.3,2.8,5.1,1.5,Iris-virginica
```

Figure 5.2: An example of the UniFeat format of a dataset.

In addition to the dataset files, a separate file that contains the values of the class feature must be imported into the tool. Each value of the class feature is presented in a row. For example, for the dataset in Figure 5.2, the class label file contains three rows, each representing a class label, including *Iris-setosa*, *Iris-versicolor*, and *Iris-virginica*. It should be noted that rows of the class label file must be compatible with the class

Table 5.1: Filter-based feature selection methods in the UniFeat repository.

Method	**Supervised/Unsupervised**	**Multivariate/Univariate**
Information gain [23]	Supervised	Univariate
Gain ratio [23]	Supervised	Univariate
Symmetrical uncertainty [21]	Supervised	Univariate
Fisher score [8]	Supervised	Univariate
Gini index [30]	Supervised	Univariate
mRMR [27]	Supervised	Multivariate
Laplacian score [13]	Supervised & unsupervised	Univariate
RRFS [6]	Supervised & unsupervised	Multivariate
Term variance [35]	Unsupervised	Univariate
Mutual correlation [11]	Unsupervised	Multivariate
RSM [19]	Unsupervised	Multivariate
UFSACO [33]	Unsupervised	Multivariate
RRFSACO_1 [32]	Unsupervised	Multivariate
RRFSACO_2 [32]	Unsupervised	Multivariate
IRRFSACO_1 [32]	Unsupervised	Multivariate
IRRFSACO_2 [32]	Unsupervised	Multivariate
MGSACO [34]	Unsupervised	Multivariate

feature values in the dataset file.

A list of some benchmark datasets from different sources that were converted to UniFeat format is available on the project website[6].

5.1.2 Choosing a feature selection method

In the "Feature selection approaches" panel, you can simply access to the well-known and state-of-the-art feature selection methods in the literature. In this panel, there are three tabs corresponding to the different feature selection approaches including the "Filter," "Wrapper," and "Embedded" tabs.

In the "Filter," "Wrapper," and "Embedded" tabs, the UniFeat repository has involved the feature selection methods, the details of which are listed in Tables 5.1 to 5.3, respectively.

Some feature selection methods have adjustable parameters that need to be set. The "More option..." button is provided in the tool to set these parameters. An example is presented in Figure 5.3 for the Laplacian score method. It should be noted that to prevent unwanted errors, the tool automatically checks the values. In other words, when the value of a parameter is empty or incorrect, a star symbol '*' immediately appears in front of the parameter to alert the user. In Figure 5.3, this issue arises for the "k-nearest neighbor" parameter.

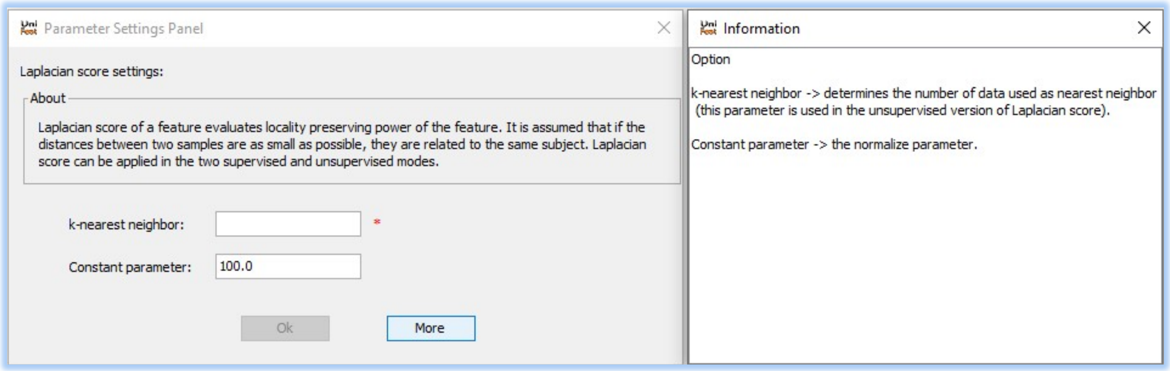
[6]https://unifeat.github.io/datasets.html

Table 5.2: Wrapper-based feature selection methods in the UniFeat repository.

Method	Supervised/Unsupervised
Binary particle swarm optimization (BPSO) [38]	Supervised
Continuous particle swarm optimization (CPSO) [38]	Supervised
Particle swarm optimization version 4-2 (PSO(4-2)) [39]	Supervised
HPSO-LS [24]	Supervised
Simple GA [15]	Supervised
HGAFS [16]	Supervised
Optimal ACO [1]	Supervised

Table 5.3: Embedded-based feature selection methods in the UniFeat repository.

Method	Supervised/Unsupervised
Decision tree based method [20]	Supervised
Random forest [3]	Supervised
SVM_RFE [10]	Supervised
MSVM_RFE [17]	Supervised
OVO_SVM_RFE [5]	Supervised
OVA_SVM_RFE [5]	Supervised



Laplacian score settings:

About

Laplacian score of a feature evaluates locality preserving power of the feature. It is assumed that if the distances between two samples are as small as possible, they are related to the same subject. Laplacian score can be applied in the two supervised and unsupervised modes.

k-nearest neighbor: *

Constant parameter: 100.0

Option

k-nearest neighbor -> determines the number of data used as nearest neighbor (this parameter is used in the unsupervised version of Laplacian score).

Constant parameter -> the normalize parameter.

Figure 5.3: An example of the parameter settings panel.

5.1.3 Feature subset sizes

In some feature selection methods, the number of selected features is a parameter that needs to be set. The "Numbers of selected features" panel is designed to set this parameter. Therefore, users are able to enter the different numbers of selected features altogether. These values must be separated by commas. Figure 5.1 shows an example of how 5, 10, 15, and 20 features should be selected by the given feature selection method.

5.1.4 Selecting classifier

In the “Classifier” panel, you can select a classifier for evaluating the subsets of features chosen by a given feature selection method. Four frequently used classifiers, including support vector machine (SVM) [10], decision tree (DT) [28], naïve Bayes (NB) [35], and k-nearest neighbors (KNN) [35], are provided to UniFeat induced from the Weka software package [12]. Also, the “More option...” button is embedded in this panel to adjust the parameters of the classifiers.

5.1.5 Run configuration

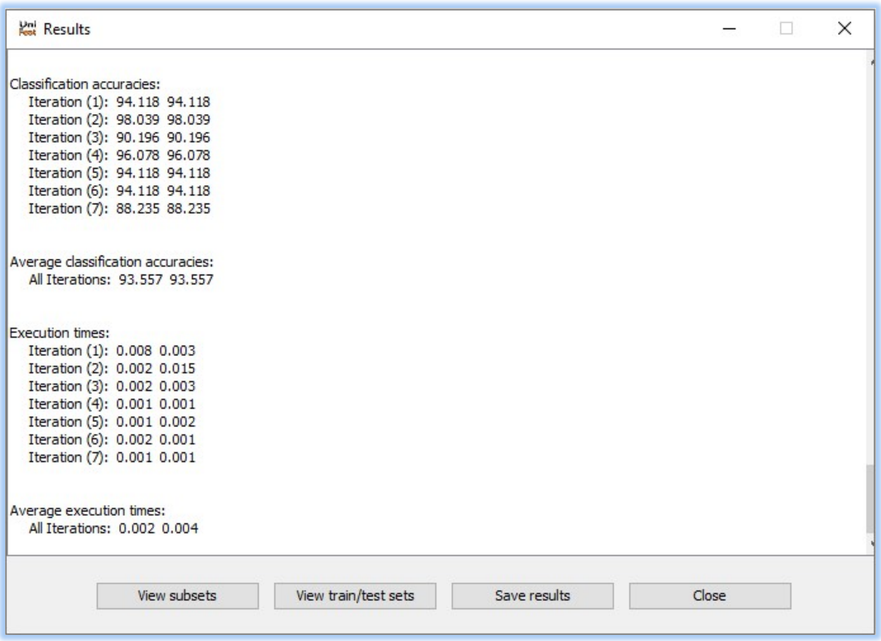
The “Run configuration” panel is designed for two purposes. (1) While input datasets are divided randomly into the training and test sets, the division process should repeat several times. This process reduces the effect of the random nature of the dataset and improves the estimation of the performance of a feature selection method. (2) Some feature selection methods embed randomness into their search processes and thus provide stable results when they run several times independently.

Finally, you can click on the “Start” button to start the feature selection process. If the user provides all the requirements of the tool with form filling, the resulting interface will be shown. In this interface, some necessary information will be reported to the user. This includes some information about the dataset, weights of features, classification accuracies, execution times, and subsets of selected features in each iteration.

Figure 5.4 shows an example of the output results generated by the tool. From Figure 5.4, it is clear that two different subsets of features are selected, and the method has been run for seven independent iterations. Note that each column corresponds to a specific subset of selected features.

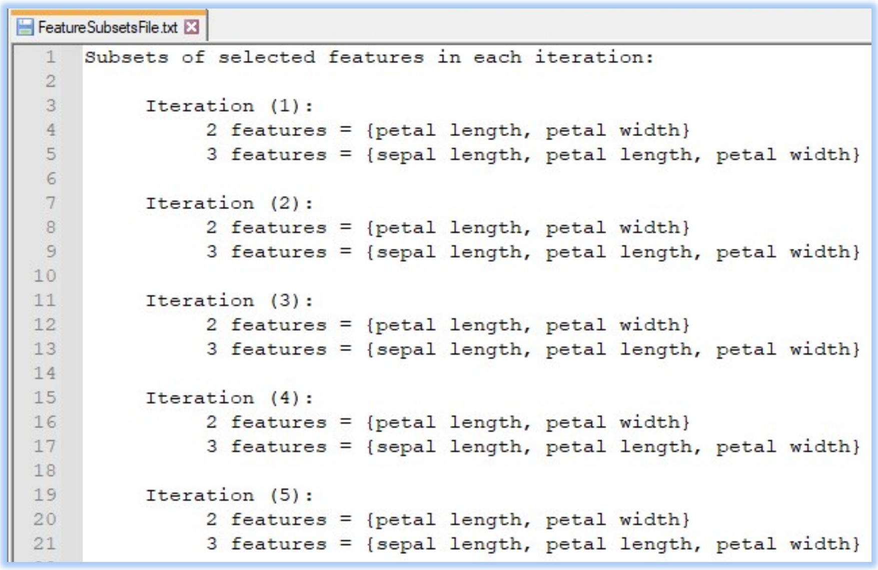
Three buttons are embedded in the resulting interface, each of which is described in the following:

1. “View subsets” button: by clicking this button, you can see the different subsets of selected features obtained by a given feature selection method in each iteration. An example of this issue is shown in Figure 5.5.
2. “View training/test sets” button: by clicking this button, you can see the two different folders with the names CSV and ARFF. The reduced datasets based on different subsets of selected features in each iteration are saved in these folders. The ARFF folder represents the reduced dataset in the attribute-relation file format (i.e., ARFF, which is the standard format of the Weka software), and the CSV folder represents the reduced datasets in the comma-delimited format (i.e., CSV file format). Each file in these folders is saved with the format “name[i-j].format”, where:
   - name: is the type of dataset with two different values: trainSet and testSet;
   - i: represents the *i*-th iteration of the tool;
   - j: shows the number of selected features;



```
Classification accuracies:
   Iteration (1):  94.118  94.118
   Iteration (2):  98.039  98.039
   Iteration (3):  90.196  90.196
   Iteration (4):  96.078  96.078
   Iteration (5):  94.118  94.118
   Iteration (6):  94.118  94.118
   Iteration (7):  88.235  88.235


Average classification accuracies:
   All Iterations:  93.557  93.557


Execution times:
   Iteration (1):  0.008  0.003
   Iteration (2):  0.002  0.015
   Iteration (3):  0.002  0.003
   Iteration (4):  0.001  0.001
   Iteration (5):  0.001  0.002
   Iteration (6):  0.002  0.001
   Iteration (7):  0.001  0.001


Average execution times:
   All Iterations:  0.002  0.004
```

Figure 5.4: An example of the resulting interface.



```
Subsets of selected features in each iteration:

    Iteration (1):
        2 features = {petal length, petal width}
        3 features = {sepal length, petal length, petal width}

    Iteration (2):
        2 features = {petal length, petal width}
        3 features = {sepal length, petal length, petal width}

    Iteration (3):
        2 features = {petal length, petal width}
        3 features = {sepal length, petal length, petal width}

    Iteration (4):
        2 features = {petal length, petal width}
        3 features = {sepal length, petal length, petal width}

    Iteration (5):
        2 features = {petal length, petal width}
        3 features = {sepal length, petal length, petal width}
```

Figure 5.5: An example of feature subsets file.

- format: illustrates the type of file format with two different values: arff and csv.

For example, the "testSet[1-5].arff" file shows the reduced test set file obtained by a given feature selection method from the first iteration based on five selected features with the ARFF format.
These reduced datasets can be easily used to compare fairly feature selection meth-

ods available in the UniFeat and any other feature selection methods implemented in the different software packages.

3. "Save results" button: You can save all information from the resulting interface as a text file by clicking this button.

5.2 Menu items description

Four different menu items correspond to the tasks of the UniFeat tool. Each of these menus is described in the following sections.

5.2.1 Preprocessing of the data

UniFeat supports only a specific dataset format described in Section 5.1.1; thus, a simple preprocessing panel is provided to help users import datasets from different sources and convert them into UniFeat format. Using the "File" → "Preprocess" menu in UniFeat, you can import a dataset and convert it to the correct format. The preprocessing panel is presented in Figure 5.6.

First, you should select a delimiter of the input dataset from the "Delimiter" panel, and then you can perform the two following optional operations over the dataset:

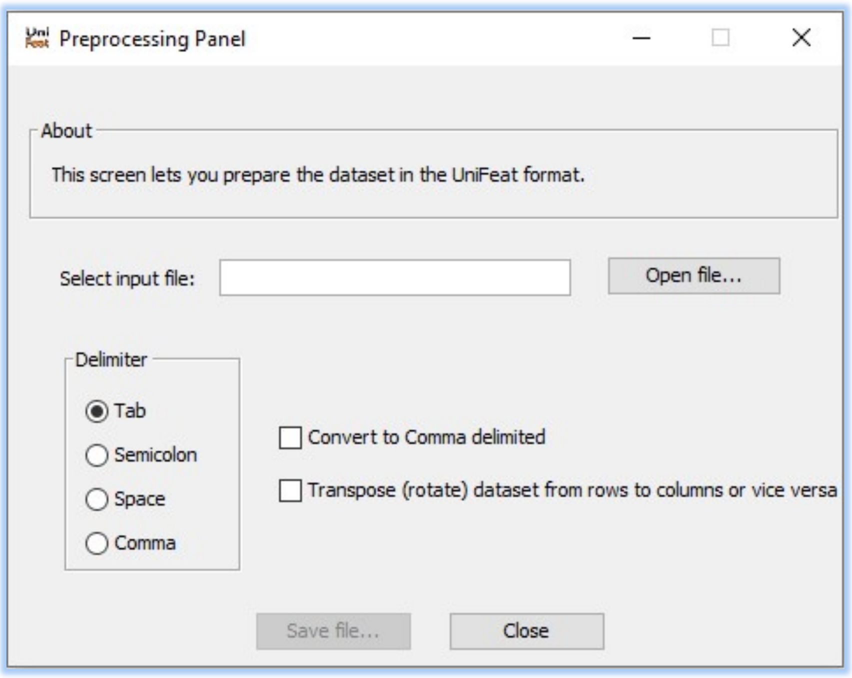
1. "Convert to Comma delimited": if you select this item, the current delimiter of the data is changed to the comma-delimited.
2. "Transpose (rotate) dataset from rows to columns or vice versa": some of the datasets have been presented so that the columns of the data show the samples, and the rows describe a vector of feature values corresponding to the samples. If you select this item, the dataset is rotated from rows to columns or vice versa.

5.2.2 Drawing a diagram

Visualizing the results obtained by UniFeat in the form of diagrams can help users obtain better interpretations. After the feature selection process is done, you can see three diagrams, including execution time, accuracy, and error rate, accessible through the "Diagram" menu. Moreover, the values of the results in each iteration and average values in all iterations can be reported in these diagrams. Figure 5.7 shows an example of the tool's execution time and classification accuracy diagrams. As shown in Figure 5.7, users can save diagrams in a *png* image format to facilitate reporting the results. This option is available in the "File" menu.

5.2.3 Analyzing the results

To show that the experimental results are statistically significant, the Friedman test [7] is currently provided in the UniFeat tool to analyze the results. The Friedman test is a non-parametric test used to measure the statistical differences of methods over multiple datasets. To apply this test, first of all, you should prepare a file as follows:



Preprocessing Panel

About

This screen lets you prepare the dataset in the UniFeat format.

Select input file: [ ] Open file...

Delimiter

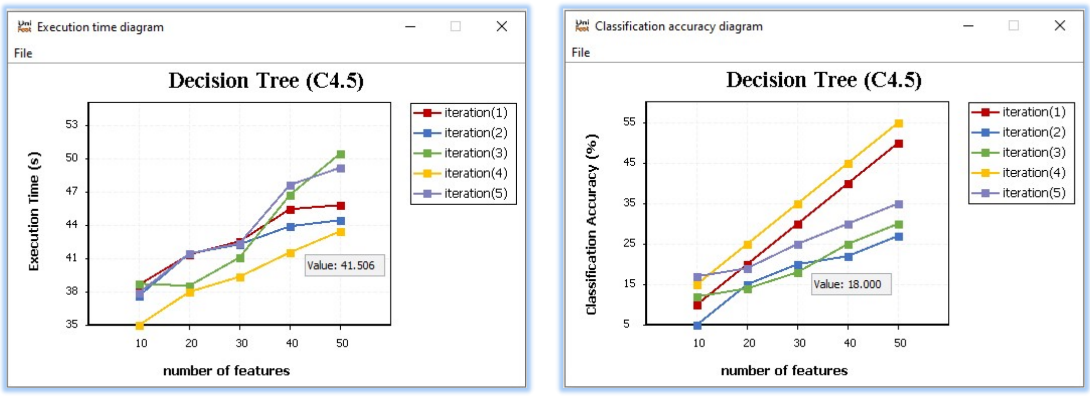
[✓ Tab]
Semicolon
Space
Comma

☐ Convert to Comma delimited

☐ Transpose (rotate) dataset from rows to columns or vice versa

Save file... Close

Figure 5.6: The preprocessing panel.



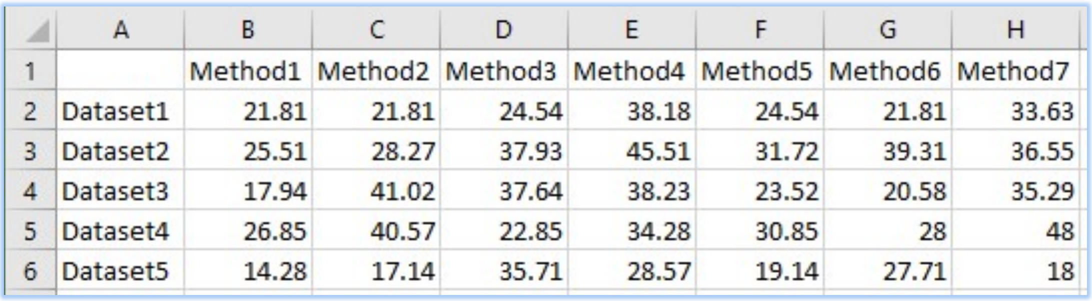
(a) Execution time diagram. (b) Classification accuracy diagram.

Figure 5.7: Diagrams of UniFeat.

- The first row of the file must have the names of methods.
- The next rows contain all the values, with each row corresponding to the results of the methods on a dataset. Each row starts with the name of a dataset, and then the results of each method are presented, separated by commas.

You can easily import this file into the UniFeat tool as a CSV file. Figure 5.8 shows an example of this file in a spreadsheet. In Figure 5.8, the input file contains the classification error rates obtained by seven methods over five different datasets.

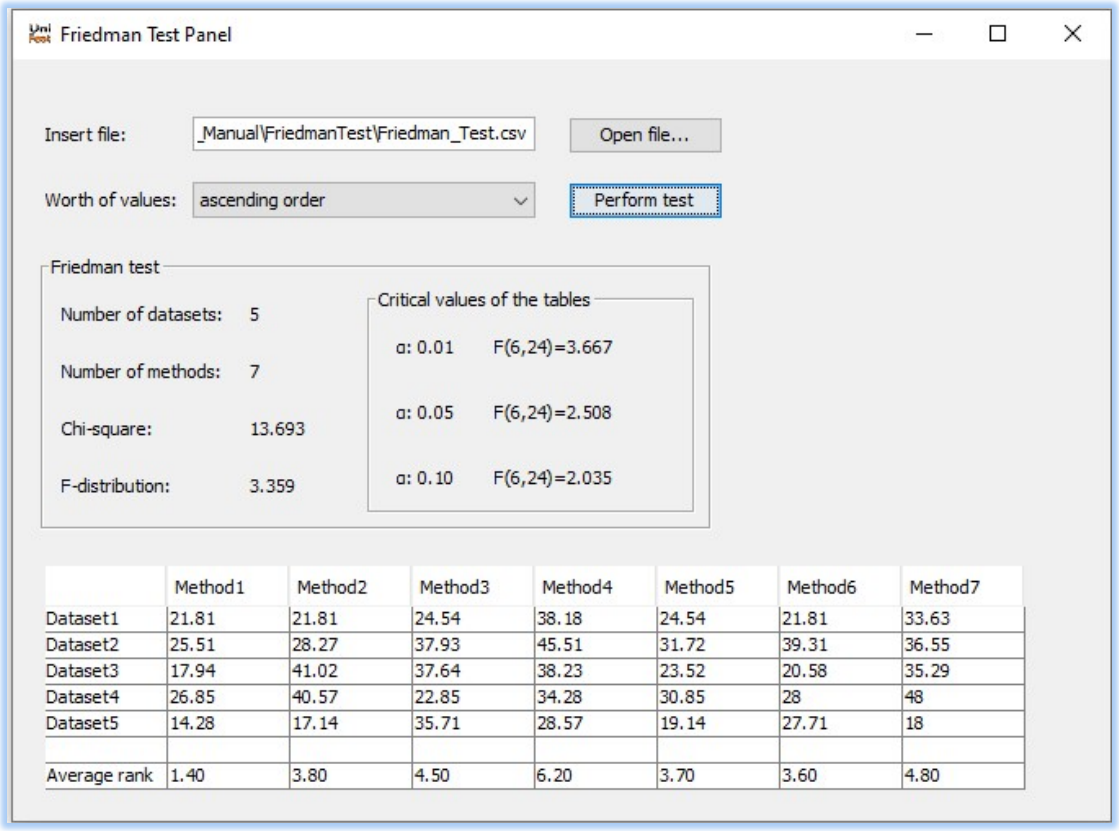
After preparing the results file, you can open the Friedman test panel, import the file,



	A	B	C	D	E	F	G	H
1		Method1	Method2	Method3	Method4	Method5	Method6	Method7
2	Dataset1	21.81	21.81	24.54	38.18	24.54	21.81	33.63
3	Dataset2	25.51	28.27	37.93	45.51	31.72	39.31	36.55
4	Dataset3	17.94	41.02	37.64	38.23	23.52	20.58	35.29
5	Dataset4	26.85	40.57	22.85	34.28	30.85	28	48
6	Dataset5	14.28	17.14	35.71	28.57	19.14	27.71	18

Figure 5.8: An example of the result values in a spreadsheet.

and perform the test on the input file using the "Analyze" → "Friedman test" menu in the UniFeat tool. The Friedman test panel is presented in Figure 5.9.



Friedman Test Panel

Insert file: _Manual\FriedmanTest\Friedman_Test.csv

Worth of values: ascending order

Friedman test

Number of datasets: 5

Number of methods: 7

Chi-square: 13.693

F-distribution: 3.359

Critical values of the tables

α: 0.01 F(6,24)=3.667

α: 0.05 F(6,24)=2.508

α: 0.10 F(6,24)=2.035

	Method1	Method2	Method3	Method4	Method5	Method6	Method7
Dataset1	21.81	21.81	24.54	38.18	24.54	21.81	33.63
Dataset2	25.51	28.27	37.93	45.51	31.72	39.31	36.55
Dataset3	17.94	41.02	37.64	38.23	23.52	20.58	35.29
Dataset4	26.85	40.57	22.85	34.28	30.85	28	48
Dataset5	14.28	17.14	35.71	28.57	19.14	27.71	18
Average rank	1.40	3.80	4.50	6.20	3.70	3.60	4.80

Figure 5.9: The Friedman test panel.

"Worth of values" in the Friedman test panel allows the user to select the worth of values in the file. If "ascending order" is specified, then the tool associates the best rank to the method with the lowest value; otherwise, in the case of "descending order," the tool associates the best rank to the method with the highest value.

The Friedman test panel provides some helpful information, such as the average values of each method over all datasets, Chi-square and F-distribution values, and critical values of the table based on various significant levels (i.e., α parameter).

5.2.4 Help file

By using the "Help" menu in UniFeat, you can access the tool's user manual.

6 Using UniFeat as a library

The easiest way to use UniFeat is through its graphical user interface. Sometimes you use feature selection as a part of your methods, and you prefer to embed the feature selection methods in your Java codes. Therefore, a question about the UniFeat tool has remained: "how to use the UniFeat tool as a jar file in your own Java codes?" In this section, we will describe this issue.

An example is presented in Appendix A to clarify how to read a dataset, call a feature selection method, and obtain the results from your own Java codes. The codes required to use the UniFeat as a jar file are explained in the following sections.

6.1 Reading dataset files

First, you should load all dataset files for future purposes. The input files must be prepared in the UniFeat format for the tool described in Section 5.1.1. If only one dataset file is available, you can easily import the codes in Figure 6.1a in your own Java code. In Figure 6.1a, *path1* is the dataset's path, and *path2* is the path of the class labels file. Both *path1* and *path2* are string values.

On the other hand, if the dataset file is originally divided into training and test sets, you can easily use the codes presented in Figure 6.1b. In Figure 6.1b, *path1* is the training set's path, *path2* is the test set's path, and *path3* is the path of the class labels file. *path1*, *path2*, and *path3* are string values.

```java
import unifeat.dataset.DatasetInfo;
...
DatasetInfo data = new DatasetInfo();
data.preProcessing(path1,path2);
```

(a) One file of the dataset.

```java
import unifeat.dataset.DatasetInfo;
...
DatasetInfo data = new DatasetInfo();
data.preProcessing(path1,path2,path3);
```

(b) Training and test files.

Figure 6.1: Source codes for reading the dataset.

6.2 Performing feature selection

After reading the dataset file, you can perform a given feature selection method based on the input dataset. The general interface of the feature selection methods currently available in the UniFeat tool can be considered in Figure 6.2.

From Figure 6.2 the following points deserve attention:

1. You can load the dataset for performing the feature selection process in two ways: (a) read the dataset as described in Section 6.1 and then use the "**loadDataSet(DatasetInfo ob)**" code, or (b) you can prepare the dataset as a matrix of double values without the names of features in the first row. Figure 6.3 shows the values of the dataset illustrated in Figure 5.2. In Figure 6.3, the data labels have

```java
public interface featureSelection {
    public void loadDataSet(DatasetInfo ob);
    public void loadDataSet(double[][] data, int numFeat, int numClasses);
    public void evaluateFeatures();
    public int[] getSelectedFeatureSubset();
    public double[] getFeatureValues();
    public String validate();
}
```

Figure 6.2: General interface of the feature selection methods.

been changed (i.e., *Iris-setosa* $\rightarrow$ 0, *Iris-versicolor* $\rightarrow$ 1, and *Iris-virginica* $\rightarrow$ 2). Then the "**loadDataSet(double[][] data, int numFeat, int numClasses)**" code is used where *numFeat* is the number of features and *numClasses* is the number of classes in the dataset.

2. The "**evaluateFeatures()**" function performs a given feature selection method over the input dataset.
3. The "**getSelectedFeatureSubset()**" function returns a subset of features selected by a given feature selection method.
4. The "**getFeatureValues()**" function is used to obtain the weights of features if the method gives weights of features individually and ranks them based on their relevance (i.e., feature weighting methods); otherwise, these values do not exist.
5. The "**validate()**" function is used to verify the validity of user input values. This method returns an empty string if all the input values are correct; otherwise, an error message is demonstrated to the user.

```java
double[][] data = { {4.3, 3, 1.1, 0.1, 0},
                    {4.4, 2.9, 1.4, 0.2, 0},
                    {4.4, 3, 1.3, 0.2, 0},
                    {4.4, 3.2, 1.3, 0.2, 0},
                    {5.5, 2.3, 4, 1.3, 1},
                    {5.5, 2.4, 3.8, 1.1, 1},
                    {5.5, 2.4, 3.7, 1, 1},
                    {5.5, 2.5, 4, 1.3, 1},
                    {6.3, 3.3, 6, 2.5, 2},
                    {6.3, 2.9, 5.6, 1.8, 2},
                    {6.3, 2.7, 4.9, 1.8, 2},
                    {6.3, 2.8, 5.1, 1.5, 2}};
```

Figure 6.3: An example of the data as a matrix.

For example, suppose we want to use information gain as a feature selection method. The required code is presented in Figure 6.4. In Figure 6.4, the *sizeSelectedFeatureSubset* parameter determines the number of features selected by the method, and the *data* is the input dataset obtained from Section 6.1. Also, the *message* keeps the possible error message from user input values, the *subset* supports the subset of features selected by information gain, and *computeValues* holds the information gain values of each feature.

```java
import unifeat.featureSelection.filter.supervised.InformationGain;
...
    InformationGain method = new InformationGain(sizeSelectedFeatureSubset);
    method.loadDataSet(data);
    String message = method.validate();
    if (!message.isEmpty()) {
        System.out.print("Error!\n " + message);
    } else {
        method.evaluateFeatures();
        int[] subset = method.getSelectedFeatureSubset();
        double[] computeValues = method.getFeatureValues();
    }
```

Figure 6.4: Source codes for performing feature selection using information gain.

6.3 Creating reduced datasets

When the feature selection process has been done, you can create reduced datasets based on the subset of selected features in the CSV or ARFF formats.

If you want to create training and test sets in CSV or ARFF file formats based on the subset of selected features (i.e., the *subset* in Figure 6.4), you can easily embed the codes in Figure 6.5 in your own Java code. In Figure 6.5, *newPathTrainCSV* is a path for the training set in CSV format, *newPathTestCSV* is a path for the test set in CSV format, *newPathTrainARFF* is a path for the training set in ARFF format, *newPathTestARFF* is a path for the test set in ARFF format, and some temporary files will be created in *tempPath*. Also, *newPathTrainCSV*, *newPathTestCSV*, *newPathTrainARFF*, *newPathTestARFF*, and *tempPath* are string values. Furthermore, the *sizeSelectedFeatureSubset* parameter determines the number of features selected by the method. This code is used when the dataset files are loaded in the way explained in Figure 6.1.

```java
import unifeat.dataset.DatasetInfo;
import unifeat.util.FileFunc;
...
    FileFunc.createCSVFile(data.getTrainSet(), subset, newPathTrainCSV,
            data.getNameFeatures(), data.getClassLabel());
    FileFunc.createCSVFile(data.getTestSet(), subset, newPathTestCSV,
            data.getNameFeatures(), data.getClassLabel());
    FileFunc.convertCSVtoARFF(newPathTrainCSV, newPathTrainARFF, tempPath,
            sizeSelectedFeatureSubset, data);
    FileFunc.convertCSVtoARFF(newPathTestCSV, newPathTestARFF, tempPath,
            sizeSelectedFeatureSubset, data);
```

Figure 6.5: Source codes for creating the CSV and ARFF files from the training and test sets based on Figure 6.1.

On the other hand, if the dataset is loaded in the form shown in Figure 6.3, and you want to create the CSV or ARFF files, you can easily embed the codes in Figure 6.6 in your own Java code. In Figure 6.6, *newPathDataCSV* is a path for the dataset in CSV format, *newPathDataARFF* is a path for the dataset in ARFF format, some temporary files will be created in *tempPath*, *FeatureNames* is an array of strings that presents the names of features, and *classNames* is an array of strings that presents the names of classes. Also, *numFeature* is the number of original features, and *numClass* is the number of classes in the dataset.

```
import unifeat.dataset.DatasetInfo;
import unifeat.util.FileFunc;
...
    FileFunc.createCSVFile(data, subset, newPathDataCSV, FeatureNames,
            classNames);
    FileFunc.convertCSVtoARFF(newPathDataCSV, newPathDataARFF, tempPath,
            sizeSelectedFeatureSubset, numFeature, FeatureNames, numClass,
            classNames);
```

Figure 6.6: Source codes for creating the CSV and ARFF files from the input array of the dataset.

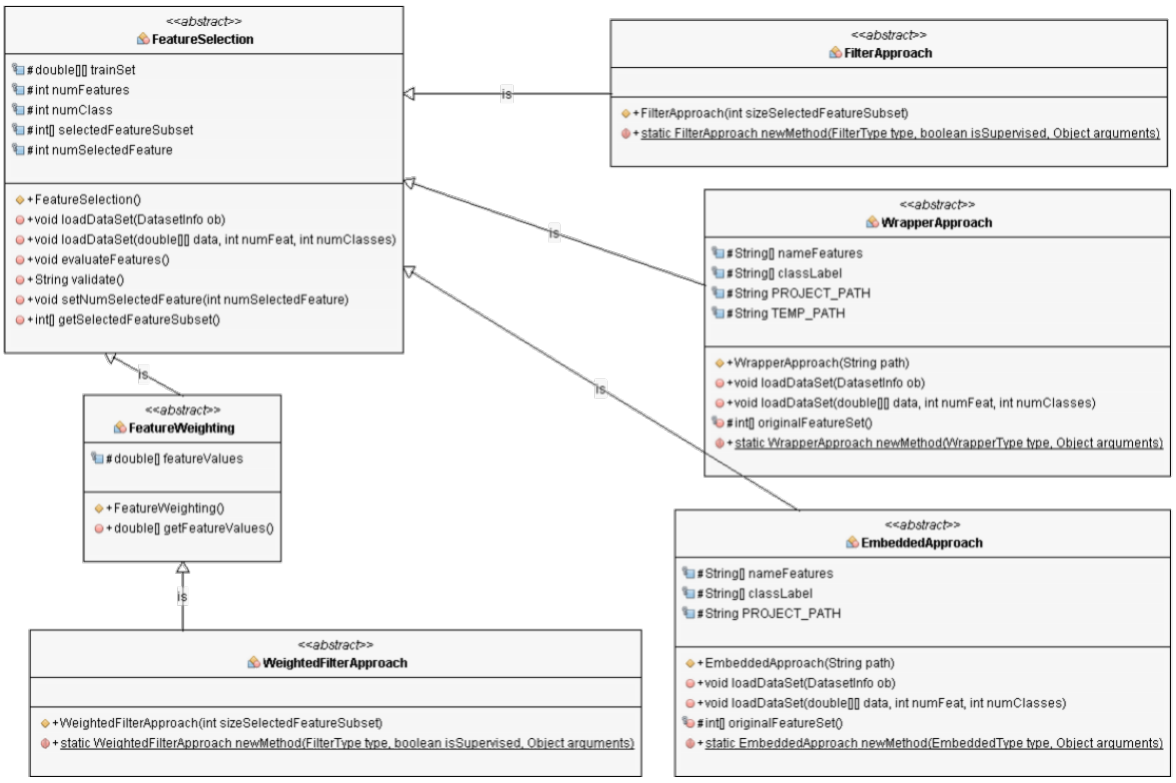
It should be noted that you can directly use the Weka software [12] to create the ARFF files based on the created CSV files.

7 Extending UniFeat

The open-source nature and structure of UniFeat can help researchers use and modify the tool to fit their research requirements and facilitate it to share their methods with the scientific community rapidly. Therefore, another question remains about the UniFeat tool: "how can a new feature selection method be added to the tool?" In this section, we will answer this question in sufficient detail.

7.1 Adding a feature selection method to UniFeat

The unified structure of the feature selection package in UniFeat allows researchers to implement their feature selection methods via the UniFeat framework. Figure 7.1 shows the UML class diagram of the UniFeat feature selection approaches. Figure 7.1 reveals that all the feature selection approaches inherit the properties of the *FeatureSelection* abstract class. The functions provided by the *FeatureSelection* class were detailed in Section 6.2.



<<abstract>>
FeatureSelection
#double[][] trainSet
#int numFeatures
#int numClass
#int[] selectedFeatureSubset
#int numSelectedFeature
+FeatureSelection()
+void loadDataSet(DatasetInfo ob)
+void loadDataSet(double[][] data, int numFeat, int numClasses)
+void evaluateFeatures()
+String validate()
+void setNumSelectedFeature(int numSelectedFeature)
+int[] getSelectedFeatureSubset()
<<abstract>>
FilterApproach
+FilterApproach(int sizeSelectedFeatureSubset)
+static FilterApproach newMethod(FilterType type, boolean isSupervised, Object arguments)
<<abstract>>
WrapperApproach
#String[] nameFeatures
#String[] classLabel
#String PROJECT_PATH
#String TEMP_PATH
+WrapperApproach(String path)
+void loadDataSet(DatasetInfo ob)
+void loadDataSet(double[][] data, int numFeat, int numClasses)
#int[] originalFeatureSet()
+static WrapperApproach newMethod(WrapperType type, Object arguments)
<<abstract>>
FeatureWeighting
#double[] featureValues
+FeatureWeighting()
+double[] getFeatureValues()
<<abstract>>
EmbeddedApproach
#String[] nameFeatures
#String[] classLabel
#String PROJECT_PATH
+EmbeddedApproach(String path)
+void loadDataSet(DatasetInfo ob)
+void loadDataSet(double[][] data, int numFeat, int numClasses)
#int[] originalFeatureSet()
+static EmbeddedApproach newMethod(EmbeddedType type, Object arguments)
<<abstract>>
WeightedFilterApproach
+WeightedFilterApproach(int sizeSelectedFeatureSubset)
+static WeightedFilterApproach newMethod(FilterType type, boolean isSupervised, Object arguments)
is
is
is
is
is

Figure 7.1: UML class diagram of feature selection approaches in UniFeat.

The current feature selection methods in the UniFeat repository are based on the filter, wrapper, and embedded approaches. We provide a separate class for each approach due to its specific requirements. Further details about how to add a new algorithm–considering its feature selection type–are provided in the corresponding sections.

7.1.1 Adding a new filter-based method

Filter-based methods are classified into two classes: the *WeightedFilterApproach* and *FilterApproach* classes, considered for feature weighting and feature subset selection methods, correspondingly. Feature weighting methods assign weights to features individually, rank them based on their relevance, and the top k features are finally returned to form the final feature set. Information gain [23], gain ratio [23], Gini index [30], and symmetrical uncertainty [21] are well-known methods in this category. On the other hand, feature subset selection methods choose a set of features without using any ranking criterion. RRFS [6], mRMR [27], RSM [19], and UFSACO [33] are examples of this category.

The general template class for adding a new filter-based method is presented in Figure 7.2, where the following points deserve attention.

```java
import unifeat.featureSelection.filter.WeightedFilterApproach;
import unifeat.util.ArraysFunc;

public class YourMethodName extends WeightedFilterApproach {

    public YourMethodName(Object... arguments) {
        super((int)arguments[0]);
    }

    public YourMethodName(int sizeSelectedFeatureSubset) {
        super(sizeSelectedFeatureSubset);
    }

    @Override
    public void evaluateFeatures() {
        // TODO feature selection process by your method
        ArraysFunc.sortArray1D(selectedFeatureSubset, false);
    }

    @Override
    public String validate() {
        // TODO validation of user input values
        return "keep this method to return an error message if"
                + " there are any errors in input parameters";
    }
}
```

Figure 7.2: General template class for adding a new filter-based method.

1. The class of your algorithm should extend one of the *WeightedFilterApproach* and *FilterApproach* abstract classes. These two abstract classes have similar functionalities, but *WeightedFilterApproach* returns a set of features associated with weights, while the other abstract class returns only a set of features.
2. Two constructor functions are provided for each filter method. The first function

has a variable argument *Object*. The second function takes the number and types of the arguments. If your method includes several tunable parameters, you should define the parameters as inputs to this function. The first value passed to both of these functions must be an integer that determines the number of features selected by your method.

3. The "**evaluateFeatures**()" function is used to implement the body of your algorithm. Note that this function does not have any input. It takes the required values from the fields provided in the *FeatureSelection* super-class, and these fields are initialized as the "**loadDataSet**()" functions are called. The body of your algorithm should store the final feature subset in the *selectedFeatureSubset* array. This array keeps the indices of the selected features, finally used as a result of your method's implementation. Moreover, the last line of your code is "**ArraysFunc.sortArray1D**()" invoked to sort the features based on their indices. The sorted array of feature indices is required to create the reduced dataset (see Section 6.3 for further details).
4. The "**validate**()" function is used to verify the validity of user input values. If your method does not include any parameter validation, you can remove this method. An implementation of this method is presented in the *FeatureSelection* super-class, where an empty message is returned to demonstrate that there is no error in the input values.

Note that you should add the class of your method into the supervised package if your method is a supervised algorithm; otherwise, you should add it into the unsupervised package.

In the GUI, users can choose feature selection methods. In UniFeat, a specific class for each approach provides a complete list of feature selection methods. All of these classes extend the *EnumType* class. Therefore, you should add the name of your filter-based method to the *FilterType* class, which is used for all filter-based feature selection methods.

7.1.2 Adding a new wrapper-based method

The general template class for adding a new wrapper-based method is similar to filter-based methods. However, the first argument in the constructor functions should be the project's path because some temporary files will be created in this path.

Since wrapper-based methods require a given classifier to evaluate different subsets of features during the feature selection process, four well-known and frequently-used classifiers are currently collected from the Weka software [12]. UniFeat uses the training/test evaluation and k-fold cross-validation [31] techniques to evaluate feature subsets. In training/test evaluation, a reduced dataset is first created based on the selected subset of features, then assessed by applying a classifier to the reduced dataset. Note that the reduced dataset is divided into training and test sets. The training set is used to build the classifier, and the test set is employed to evaluate the performance of the selected features.

Figure 7.3 shows the declarations of the current classifiers used for training/test evaluation. In these declarations, *pathTrainData* is the path of the training set in ARFF format, and *pathTestData* is the path of the test set in ARFF format. Other arguments in each function are needed for a specific classifier. All these functions are static, which means they can be invoked directly from the *TrainTestEvaluation* class.

```java
public static Criteria SVM(String pathTrainData, String pathTestData,
                           SVMKernelType svmKernel, double c);
public static Criteria naiveBayes(String pathTrainData, String pathTestData);
public static Criteria dTree(String pathTrainData, String pathTestData,
                           double confidenceValue, int minNumSampleInLeaf);
public static Criteria kNN(String pathTrainData, String pathTestData,
                           int kNNValue);
```

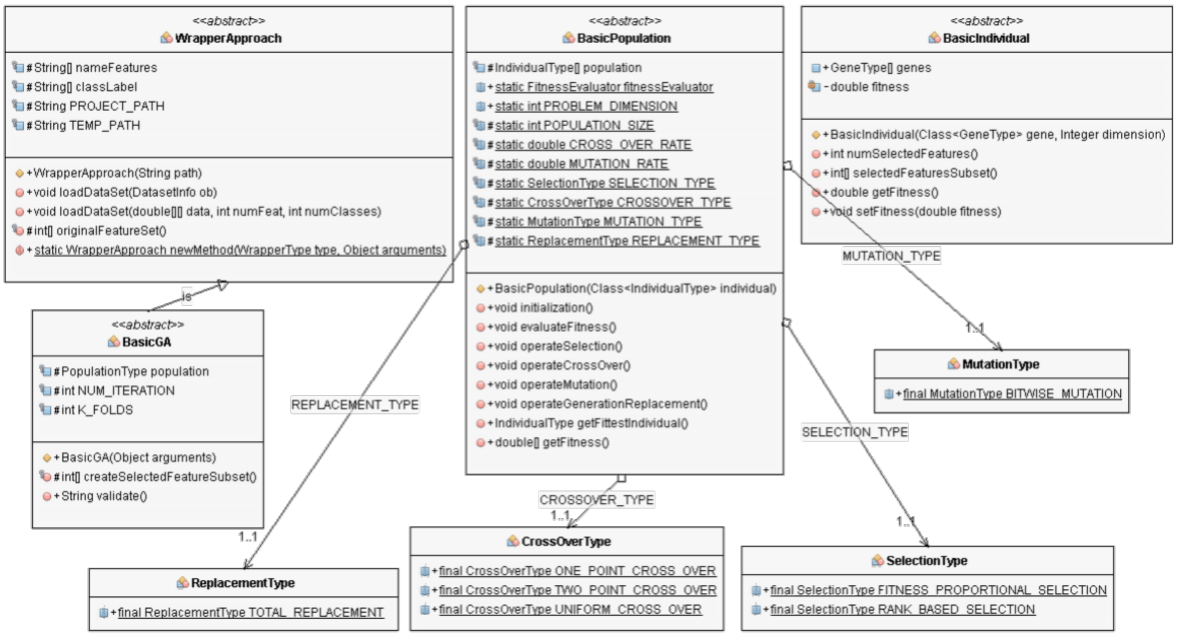
Figure 7.3: Declarations of the classifiers used for training/test evaluation in UniFeat.

In k-fold cross-validation, the dataset is split into k parts. The first k-1 parts are applied in the training process to build a classifier. At the same time, the last one is utilized in the validation process to evaluate the performance of the selected subset. Figure 7.4 shows the declarations of the current classifiers employed in k-fold cross-validation. In Figure 7.4, *pathTrainData* is the path of the training set in ARFF format. Furthermore, *kFold* is an argument for defining the number of folds. All these functions are static, which means they can be invoked directly from the *CrossValidation* class.

```java
public static Criteria SVM(String pathTrainData, SVMKernelType svmKernel,
                           double c, int kFold);
public static Criteria naiveBayes(String pathTrainData, int kFold);
public static Criteria dTree(String pathTrainData, double confidenceValue,
                           int minNumSampleInLeaf, int kFold);
public static Criteria kNN(String pathTrainData, int kNNValue, int kFold);
```

Figure 7.4: Declarations of the classifiers used for k-fold cross-validation in UniFeat.

Recently, population-based methods have attracted a lot of attention. Most of them belong to the wrapper approach. These methods consider the interaction between subsets of features, and they show higher performance than filter-based methods. The three most popular methods, including Genetic Algorithm (GA), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO), are implemented in UniFeat. The simple implementation of these algorithms in UniFeat helps researchers easily have the structures inherited in their methods to develop. For example, Figure 7.5 provides the basic structure of GA implemented in UniFeat. In this structure, we have provided three abstract classes, including *BasicIndividual*, *BasicPopulation*, and *BasicGA*, which can be used in any GA-based feature selection method. The *BasicIndividual* class is employed to represent an individual, and the *BasicPopulation* class is used to create a population of individuals and apply genetic operations. Finally, *BasicGA* is the main class utilized for iteration of the algorithm an allowed number of times. Moreover, *BasicGA* class inherits the properties of the *WrapperArpproach* class. It is clear from Figure 7.5 that the essential genetic operators, including crossover, mutation, selection, and replacement, have been implemented in UniFeat. As frequently-used operators, they can be invoked from your method.



<<abstract>> WrapperApproach
#String[] nameFeatures
#String[] classLabel
#String PROJECT_PATH
#String TEMP_PATH
+WrapperApproach(String path)
+void loadDataSet(DatasetInfo ob)
+void loadDataSet(double[][] data, int numFeat, int numClasses)
#int[] originalFeatureSet()
+static WrapperApproach newMethod(WrapperType type, Object arguments)
<<abstract>> BasicPopulation
#IndividualType[] population
+static FitnessEvaluator fitnessEvaluator
+static int PROBLEM_DIMENSION
#static int POPULATION_SIZE
#static double CROSS_OVER_RATE
#static double MUTATION_RATE
#static SelectionType SELECTION_TYPE
#static CrossOverType CROSSOVER_TYPE
#static MutationType MUTATION_TYPE
#static ReplacementType REPLACEMENT_TYPE
+BasicPopulation(Class<IndividualType> individual)
+void initialization()
+void evaluateFitness()
+void operateSelection()
+void operateCrossOver()
+void operateMutation()
+void operateGenerationReplacement()
+IndividualType getFittestIndividual()
+double[] getFitness()
<<abstract>> BasicIndividual
+GeneType[] genes
-double fitness
+BasicIndividual(Class<GeneType> gene, Integer dimension)
+int numSelectedFeatures()
+int[] selectedFeaturesSubset()
+double getFitness()
+void setFitness(double fitness)
is
<<abstract>> BasicGA
#PopulationType population
#int NUM_ITERATION
#int K_FOLDS
+BasicGA(Object arguments)
#int[] createSelectedFeatureSubset()
+String validate()
REPLACEMENT_TYPE
MUTATION_TYPE
1..1
MutationType
+final MutationType BITWISE_MUTATION
SELECTION_TYPE
CROSSOVER_TYPE
1..1
1..1
1..1
ReplacementType
+final ReplacementType TOTAL_REPLACEMENT
CrossOverType
+final CrossOverType ONE_POINT_CROSS_OVER
+final CrossOverType TWO_POINT_CROSS_OVER
+final CrossOverType UNIFORM_CROSS_OVER
SelectionType
+final SelectionType FITNESS_PROPORTIONAL_SELECTION
+final SelectionType RANK_BASED_SELECTION

Figure 7.5: UML class diagram of the genetic algorithm in UniFeat.

7.1.3 Adding a new embedded-based method

The general template class for adding a new embedded method is similar to wrapper-based methods. In embedded-based methods, a given classifier is trained by an original feature set, and the obtained results are used to specify the relevance of each feature. Therefore, the functions shown in Figures 7.3 and 7.4 can be used in this approach.

SVM and DT are common classifiers for embedded-based methods implemented in UniFeat. For instance, Figure 7.6 shows the abstract classes of SVM-based methods. In Figure 7.6, there are two main functions "**buildSVM_OneAgainstOne**()" and "**buildSVM_OneAgainstRest**()". In the first function, there is a binary SVM for each pair of sample classes to separate the sample of one class from that of the other. In the second function, however, there is a binary SVM for each sample class to separate the sample of that class from that of the other classes.

7.2 Creating a parameter settings panel in UniFeat

Some feature selection methods have tunable parameters that need to be set. Further details on the parameters of different methods are included in Section 5.1.2. A simple structure has been designed for developers to create a GUI panel for setting these

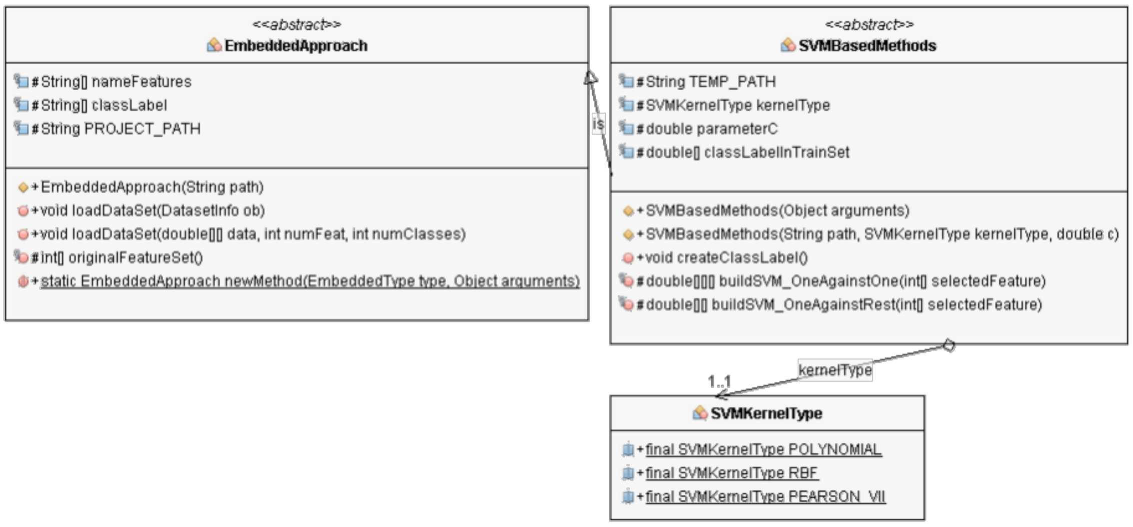

<<abstract>>
EmbeddedApproach
#String[] nameFeatures
#String[] classLabel
#String PROJECT_PATH
+EmbeddedApproach(String path)
+void loadDataSet(DatasetInfo ob)
+void loadDataSet(double[][] data, int numFeat, int numClasses)
#int[] originalFeatureSet()
+static EmbeddedApproach newMethod(EmbeddedType type, Object arguments)
is
<<abstract>>
SVMBasedMethods
#String TEMP_PATH
#SVMKernelType kernelType
#double parameterC
#double[] classLabelInTrainSet
+SVMBasedMethods(Object arguments)
+SVMBasedMethods(String path, SVMKernelType kernelType, double c)
+void createClassLabel()
#double[][][] buildSVM_OneAgainstOne(int[] selectedFeature)
#double[][] buildSVM_OneAgainstRest(int[] selectedFeature)
kernelType
1..1
SVMKernelType
+final SVMKernelType POLYNOMIAL
+final SVMKernelType RBF
+final SVMKernelType PEARSON_VII

Figure 7.6: UML class diagram of SVM in UniFeat.

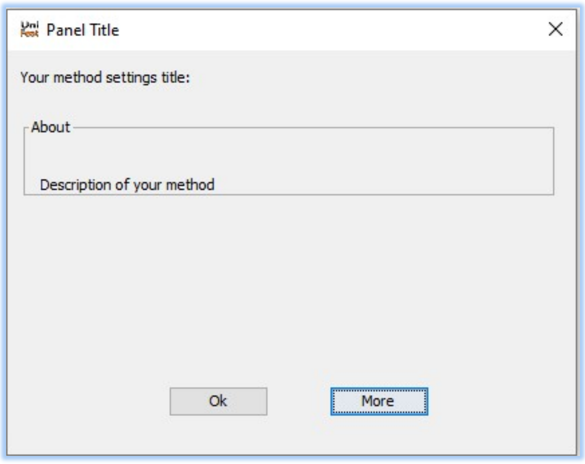
parameters in the UniFeat tool. Figure 7.7 shows a general template for creating a panel in UniFeat.

After running the code provided in Figure 7.7, you will see the general panel of parameter settings, which is illustrated in Figure 7.8a. You can change the "Panel Title," "Your method settings title," and "Description of your method" by calling the functions presented in the *ParameterPanel* super-class. Moreover, adding the desired components to the "**YourPanel**()" constructor function will be observed in the panel illustrated in Figure 7.8a. Furthermore, as seen in Figure 7.8a, if a user clicks on the "More" button, Figure 7.8b will be shown. Further information about the parameters is presented in this panel, and you can change the panel text by calling the relevant function in the *ParameterPanel* class.

After creating the panel, you should add the required code to the *MainPanel* class in the gui package. In this class, four important functions are presented: "**getFilterApproachParameters**()," "**getWeightedFilterApproachParameters**()," "**getWrapperApproachParameters**()," and "**getEmbeddedApproachParameters**()." They pass input parameters, which are obtained through GUI, to a specific method. These four functions are designed for different feature selection approaches.

```java
import unifeat.gui.ParameterPanel;
import java.awt.Dialog;
import java.awt.event.KeyEvent;
import javax.swing.UIManager;

public class YourPanel extends ParameterPanel {

    public YourPanel(){
        super();
    }

    @Override
    public void keyReleased(KeyEvent e) {
        //TODO action when a key has been released
    }

    public static void main(String[] arg) {
        try {
            UIManager.setLookAndFeel(
                                UIManager.getSystemLookAndFeelClassName());
            UIManager.getDefaults().put("TextArea.font",
                                UIManager.getFont("TextField.font"));
        } catch (Exception e) {
            System.out.println("Error: " + e);
        }

        YourPanel dtpanel = new YourPanel();
        Dialog dlg = new Dialog(dtpanel);
        dtpanel.setVisible(true);
    }
}
```

Figure 7.7: General template class for creating a GUI panel in UniFeat.

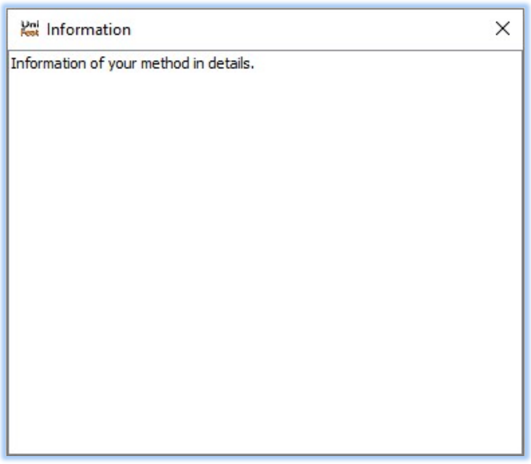


Your method settings title:

About

Description of your method

(a) Parameter settings panel.



Information of your method in details.

(b) Information about the parameters.

Figure 7.8: GUI panel in UniFeat.

References

[1] M. H. Aghdam, N. Ghasem-Aghaee, and M. E. Basiri. Text feature selection using ant colony optimization. *Expert Systems with Applications*, 36(3):6843–6853, 2009.

[2] V. Bolón-Canedo, N. Sánchez-Maroño, A. Alonso-Betanzos, J. M. Benítez, and F. Herrera. A review of microarray datasets and applied feature selection methods. *Information Sciences*, 282:111–135, 2014.

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] G. Brown, A. Pocock, M.-J. Zhao, and M. Lujan. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. *Journal of Machine Learning Research*, 13(1):27–66, 2012.

[5] K.-B. Duan, J. C. Rajapakse, and M. N. Nguyen. One-versus-one and one-versus-all multiclass svm-rfe for gene selection in cancer classification. In *Evolutionary Computation,Machine Learning and Data Mining in Bioinformatics*, pages 47–56. Springer Berlin Heidelberg, 2007.

[6] A. J. Ferreira and M. A. T. Figueiredo. An unsupervised approach to feature discretization and selection. *Pattern Recognition*, 45(9):3048–3060, 2012.

[7] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200): 675–701, 1937.

[8] Q. Gu, Z. Li, and J. Han. Generalized fisher score for feature selection. In *International Conference on Uncertainty in Artificial Intelligence*, 2011.

[9] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[10] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

[11] M. Haindl, P. Somol, D. Ververidis, and C. Kotropoulos. *Feature Selection Based on Mutual Correlation*, volume 4225 of *Lecture Notes in Computer Science*, book section 59, pages 569–577. Springer Berlin Heidelberg, 2006.

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[13] X. He, D. Cai, and P. Niyogi. Laplacian score for feature selection. *Advances in Neural Information Processing Systems*, 18, 2005.

[14] M. Hofmann and R. Klinkenberg. *RapidMiner: Data Mining Use Cases and Business Analytics Applications.* Chapman & Hall/CRC, 2013.

[15] F. T. Hussein. *Genetic algorithm for feature selection and weighting for off-line character recognition.* Thesis, University of British Columbia, 2002.

[16] M. M. Kabir, M. Shahjahan, and K. Murase. A new local search based hybrid genetic algorithm for feature selection. *Neurocomputing*, 74(17):2914–2928, 2011.

[17] D. Kai-Bo, J. C. Rajapakse, W. Haiying, and F. Azuaje. Multiple svm-rfe for gene selection in cancer classification with expression data. *IEEE Transactions on NanoBioscience*, 4(3):228–234, 2005.

[18] R. Kohavi, G. John, R. Long, D. Manley, and K. Pfleger. Mlc++: a machine learning library in c++. In *Proceedings Sixth International Conference on Tools with Artificial Intelligence. TAI 94*, pages 740–743, 1994.

[19] C. Lai, M. J. T. Reinders, and L. Wessels. Random subspace method for multivariate feature selection. *Pattern Recognition Letters*, 27(10):1067–1076, 2006.

[20] T. N. Lal, O. Chapelle, J. Weston, and A. Elisseeff. *Embedded Methods*, pages 137–165. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[21] H. Liu and H. Motoda. *Computational Methods of Feature Selection.* Chapman & Hall/CRC, 2007.

[22] H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4): 491–502, 2005.

[23] T. M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., 1997.

[24] P. Moradi and M. Gholampour. A hybrid particle swarm optimization for feature subset selection by integrating a novel local search strategy. *Applied Soft Computing*, 43:117–130, 2016.

[25] P. Moradi and M. Rostami. Integration of graph clustering with ant colony optimization for feature selection. *Knowledge-Based Systems*, 84:144–161, 2015.

[26] P. M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 26(9):917–922, 1977.

[27] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005.

[28] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[29] Y. Saeys, I. Inza, and P. Larrañaga. A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517, 2007.

[30] W. Shang, H. Huang, H. Zhu, Y. Lin, Y. Qu, and Z. Wang. A novel feature selection algorithm for text categorization. *Expert Systems with Applications*, 33(1):1–5, 2007.

[31] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974.

[32] S. Tabakhi and P. Moradi. Relevance–redundancy feature selection based on ant colony optimization. *Pattern Recognition*, 48(9):2798–2811, 2015.

[33] S. Tabakhi, P. Moradi, and F. Akhlaghian. An unsupervised feature selection algorithm based on ant colony optimization. *Engineering Applications of Artificial Intelligence*, 32(0):112–123, 2014.

[34] S. Tabakhi, A. Najafi, R. Ranjbar, and P. Moradi. Gene selection for microarray data classification using a novel ant colony optimization. *Neurocomputing*, 168: 1024–1036, 2015.

[35] S. Theodoridis and K. Koutroumbas. *Pattern Recognition.* Elsevier Science, 2008.

[36] V. Trevino and F. Falciani. Galgo: an r package for multivariate variable selection using genetic algorithms. *Bioinformatics*, 22(9):1154–1156, 2006.

[37] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011.

[38] B. Xue. *Particle Swarm Optimisation for Feature Selection in Classification.* Thesis, Victoria University of Wellington, 2014.

[39] B. Xue, M. Zhang, and W. N. Browne. Particle swarm optimisation for feature selection in classification: Novel initialisation and updating mechanisms. *Applied Soft Computing*, 18:261–276, 2014.

[40] K. Yu, W. Ding, and X. Wu. Lofs: A library of online streaming feature selection. *Knowledge-Based Systems*, 113:1–3, 2016.

A An example source code for feature selection

A simple Java program that performs feature selection using the information gain method implemented in the UniFeat tool and displays the results is presented below:

```java
import unifeat.dataset.DatasetInfo;
import unifeat.featureSelection.filter.supervised.InformationGain;
import unifeat.util.FileFunc;

public class Main {
    public static void main(String[] args) {
        //reading the datasets files
        DatasetInfo data = new DatasetInfo();
        data.preProcessing("data/trainSet.csv", "data/testSet.csv", "data/classLabels.txt");

        //printing some information of the dataset
        int sizeSelectedFeatureSubset = 2;
        System.out.println(" no. of all samples : " + data.getNumData()
                + "\n no. of samples in training set :  " + data.getNumTrainSet()
                + "\n no .of samples in test set : " + data.getNumTestSet()
                + "\n no. of features : " + data.getNumFeature()
                + "\n no. of classes : " + data.getNumClass());

        //performing the feature selection by information gain method
        InformationGain method = new InformationGain(sizeSelectedFeatureSubset);
        method.loadDataSet(data);

        String message = method.validate();
        //checking the validity of user input values
        if (!message.isEmpty()) {
            System.out.print("Error!\n  " + message);
        } else {
            method.evaluateFeatures();
            int[] subset = method.getSelectedFeatureSubset();
            double[] infoGainValues = method.getFeatureValues();

            //printing the subset of selected features
            System.out.print("\n subset of selected features: ");
            for (int i = 0; i < subset.length; i++) {
                System.out.print((subset[i] + 1) + "  ");
            }

            //printing the information gain values
            System.out.println("\n\n information gain values: ");
            for (int i = 0; i < infoGainValues.length; i++) {
                System.out.println(" " + (i + 1) + " : " + infoGainValues[i]);
            }

            //creating reduced datasets as the CSV file format
            FileFunc.createCSVFile(data.getTrainSet(), subset, "data/newTrainSet.csv",
            ↪  data.getNameFeatures(), data.getClassLabel());
            FileFunc.createCSVFile(data.getTestSet(), subset, "data/newTestSet.csv",
            ↪  data.getNameFeatures(), data.getClassLabel());

            //creating reduced datasets as the ARFF file format
            FileFunc.convertCSVtoARFF("data/newTrainSet.csv", "data/newTrainSet.arff", "data",
            ↪  sizeSelectedFeatureSubset, data);
            FileFunc.convertCSVtoARFF("data/newTestSet.csv", "data/newTestSet.arff", "data",
            ↪  sizeSelectedFeatureSubset, data);
        }
    }
}
```