Modern Implementations of Feature Selection Algorithms and Their Perspectives

Nikita Pilnenskiy, Ivan Smetannikov
ITMO University
St.Petersburg, Russia
ndpilnenskii, ismetannikov@itmo.ru

***Abstract*—In the last decade, due to the growing interest in neural networks and machine learning in general, the Python programming language became the main language for many data scientists and machine learning engineers. This rapid growth resulted into the lack of many key machine learning algorithms in the existing Python libraries. Feature selection, as one of the main fields of data preprocessing for high-dimensional, was not covered properly in Python language, although it can be widely used to improve models quality and solve some of the overfitting problems. In this paper we have performed a review of existing open-source Python feature selection libraries, made their comparison and pointed their drawbacks and after that presented our own open-source ITMO FS library. Moreover, we have added some examples on its usage and compatibility with most popular modern machine learning Python library scikit-learn and some performance tests.**

I. Introduction

One of the main problems of the last decades is the problem of information growth. Such domains as bioinformatics, social analysis, medical care and others are producing big amounts of high-dimensional data [1], [2], [3], [4]. With this kind of data the problem of "curse of dimensionality" arises. It was described in [5] and basically means that with big number of features and small number of objects your dataset becomes really too sparse for building valid models. This happens due to the fact that you have small number of object in high-dimensional data and as a result the model has too much freedom of generalisation which results into overfitting problem so the quality of the model on the new incoming data drops dramatically.

One of machine learning and data analysis main fields that is intended to solve this problem is dimensionality reduction. It is divided into two big parts, namely, feature selection and feature extraction [6]. The last one tries to reconstruct new features based on the old ones so the new space dimensionality is smaller than the old one. Of course, during this transformation, the semantics of original features is usually lost, so this approach is limited for many fields like bioinformatics or medicine. On the other hand, feature selection gets new feature set as the subset of the original one, thus saving the original features semantics [7]. Nevertheless, both of these approaches can reduce the feature set and increase the quality of the resulting models.

With the growth of the computational power the neural networks approach became really powerful in many machine learning applications. As most of the libraries designed to work with neural networks was programmed on the Python language, it became de facto the international standard for neural network research [8], [9]. As popularity of machine learning grew, more and more researchers were attracted to this field, and as neural networks was a huge attraction point for the last years, most of the modern machine learning researchers are using Python as their main programming language. These factors resulted into huge gap between Python machine learning libraries and libraries on other languages. Basically, nearly all machine learning fields that are not closely tight with neural networks are not properly covered with programming libraries in Python. In this paper we have covered main existing open-source Python feature selection libraries, showed their advantages and disadvantages and proposed our own ITMO FS [10]. Also we have performed a comparison with Arizona State University feature selection library [11] and scikit-learn feature selection module [12],this comparison is the most complete one for the Python language.

The rest of the paper is organized as follows: Section II reviews existing ways for feature selection algorithms categorization, Section III offers a survey of existing feature selection Python libraries and their analysis, Section IV contains description of proposed ITMO FS library and its comparison with libraries surveyed in previous section, Section V has some code samples for better understanding of ITMO FS library architecture and comparison of its performance with some other libraries, and Section VI contatins conclusion.

II. Background

In order to get better understanding of how the modern feature selection library should be designed and what should be included into it, we need to get better understanding of what types of feature selection algorithms are available. In this section we have presented all main categorizations of existing feature selection algorithms.

Generally speaking, the feature selection problem can be formalized as follows [13]: For a given dataset D with M objects described with feature set F, $|F| = N$ we need to find some optimal subset of features F^*, $F^* \subseteq F$ in terms of some optimization criteria C.

A. Traditional feature selection algorithms categorization

Traditionally, feature selection methods were divided into three main groups: wrapper, filter and embedded methods [14]. Wrappers are trying to build optimal feature subset by the evaluation of the quality measure Q_c for the predefined machine

learning algorithm:

$$F^* = \underset{F' \subseteq F}{argmax}\ Q_c(F')$$

, where C is a machine learning model and Q is the quality measure for the model. For this wrapper algorithms works iterative, on each step it takes some feature subset and passes it to the model and then, depending on the model quality, it decides to pick another subset or stop the process. The picking procedure and the stopping criteria are basically defining wrapper algorithm. The main problem of this approach is that too slow for high dimensionality datasets as number of possible subsets is equal to 2^N and on each step we need to build a model to evaluate the subset quality.

Embedded methods are usually using some intrinsic properties of the classifier in order to get features subset. Feature selection with random forest [15] is an illustraitive example of such approach, where out of bag error for each feature on each tree is aggregated into resulting feature scores and features that most often resulted into elimination of bad classification results. Some of these methods can work even with really high-dimensional data, but their main restricion is the model itself. Basically, features that were selected with one model, can result into bad performance if they are used for another model.

The third traditional group of feature selection algorithms are filters. Instead of evaluating the feature sets with some models, they are taking into consideration only intrinsic properties of the features themselves. If filter is not taking to consideration any dependencies between features themselves, thus assuming that they are independent, it is called univariate, otherwise – multivariate. For multivariate filters problem is stated as follows:

$$F^* = \underset{F' \subseteq F}{argmax}\ \mu(F')$$

, where μ is the feature subset quality measure. On the other hand, for the univariate filters the feature selection problem is stated without optimization. Instead of this, every feature is evaluated with feature quality measure μ (which for this case should be defined only on $f_i \in F$, but not for the whole set of features subsets) and then some cutting rule κ is applied.

B. Hybrid and ensembling feature selection algorithms

Nowadays, most scientists distinguish separate groups of feature selection algorithms: hybrids and ensembles. Basically, these types of algorithms are implementing consecutive and parallel approaches to combine feature selection algorithms.

Hybrid feature selection algorithms are trying to combine traditional approaches consecutively. This is a powerful compromise between different traditional approaches. For example, in order to select features from high-dimensional dataset, filter as a first step can be applied in order to drastically reduce feature set and after that wrapper can be applied to the output feature set to get maximum quality from the extracted features.

Alternatively, ensemble feature selection algorithms combine several feature selection algorithms in parallel in order to improve their quality or even get better stability of selected feature subsets. Ensemble feature selection algorithms work either with separetely selected features subsets or with the models, built on them [16].

C. Feature selection algorithms categorization by input data

Some of the researchers categorize feature selection algorithms depending on the input data types. Basically, all data can be divided into streaming and static data. As for streaming data, it can be divided into two big groups: data stream, when new object are added consecutively and feature stream when new features are added to the dataset.

As for static data, that is more conventional for traditional feature selection algorithms, some researchers [17] categorize them into: similarity based, information theoretical based, sparse learning based, statistical based methods and others. Similarity based methods are building an affinity matrix in order to get feature scores. As they are kind of univariate filters in traditional classification, they are not taking into corporation any model and do not handle feature redundancy. Information theoretical algorithms work the same way as similarity based ones but also utilize the concept of “feature redundancy”. Sparse learning based feature selection algorithms are embedding feature selection into a machine learning algorithms, working with weights of features. Statistical based feature selection are using statistical measures for features filtering, thus working exactly like most filtering methods in traditional interpretation. As could be seen from this categorization, it is not including any wrappers, so they are usually categorized as “other”.

III. Survey of existing Python feature selection libraries

This section contains description of existing open-source libraries and repositories with feature selection algorithms on Python.

A. Default scikit-learn feature selection

The scikit-learn library is de facto the most commonly used machine learning libraries for Python nowadays. Is is so popular, that nearly all other libraries and frameworks that were developed in recent years are scikit-learn compatible and easy to use with. Nevertheless, the feature selection module of this library [12] looks really empty as it implements only several filters and wrappers. Comparison of it contents with other libraries can be found in the Table I at SKL column.

B. Scikit-feature (Arizona State University)

Scikit-feature is a Python open-source feature selection library developed by Data Mining and Machine Learning Lab at Arizona State University [11]. Nowadays it is the biggest Python feature selection library that exists, it includes around 40 different feature selection algorithms. Moreover, it is completely scikit-learn compatible and easy to use. However, there are two main issues related to this library. Firstly, its development was stopped around two years ago, and at this stage library does still not have many feature selection algorithms, especially of hybrid and ensemble types. Secondly, it is build upon feature selection algorithms categorization by the input data. This results into some issues

of compatibility with theoretical basics of some algorithms and puts a lot of limitations on the possible development of the algorithms. As an example of such limitations we can state:

```
01 def test_filter(self):
02   data, target = self.basehock['X'],
     self.basehock['Y']
03   features = gini_index.gini_index(data,
     target)
04   k_best = lambda x, k: [keys[0]
     for keys in sorted(x.items(),
     key=lambda kv: kv[1],
     reverse=True)[:k]]
05   k_best(dict(zip([i for i in
     range(len(features))], features)),
10)
```

As could be seen from this example, at line 02 this library only extracts feature list with feature measures, but does not consider taking cutting rule as an additional input. Of course, we can implement required cutting rule directly in the code like it is presented on the line 03 and then apply it as in line 04, but this is not quite user-friendly approach, as without any template limitations during the implementation process user can incorporate many additional errors into the code. Comparison of this library contents with other libraries can be found in the Table I at ASU column.

C. Boruta methods

The Boruta methods repository [18] consists of the Python implementation of the Boruta R package that is also a scikit-learn compatible. This repository is not included in Table I as it contains Boruta methods only and nothing else, unlike other libraries.

D. MLFeatureSelection library

The MLFeatureSelection library contains some heuristic feature selection algorithms based on certain machine learning methods and their evaluation techniques, such as: sequence selection, importance selection and coherence selection. This library contains these methods only that is why it is not added to the comparison Table I. Nevertheless, it has achieved impressive results in Rong360, JData-2018 and IJCAI-2018 competitions.

E. FES book support code

This repository [19] contains sample codes for the text "Feature Engineering and Selection: A Practical Approach for Predictive Models" by Kuhn and and Johnson and published by Chapman & Hall/CRC (who holds the copyright). Most of the book contents consists of feature extraction algorithms, but the last three chapters are mainly focused on the feature selection ones, namely, RFE, ROC filter, simulated annealing and some basic implementations of genetic algorithms.

F. ReBATE algorithm

As most popular library the ReBATE repository contains several RelieF-based methods implementations that fits with scikit-learn pipeline structure. As this library also basically contains only one method, it was not included to the comparison Table I.

IV. ITMO_FS library architecture and comparison

ITMO feature selection library architecture developing was inspired by traditional feature selection categorization. So with regard to this type of categorization, it includes filters, wrappers and hybrids. Embedded methods were excluded from this implementation at the current state due to the fact that they require deep integration with machine learning models themselves. All filters generally described as shown in the Section II-A thus they need to have some feature quality measure and a cutting rule. Wrapper module contains several wrapper algorithms. Each of them takes an estimator as input parameter and improve feature set regarding to model performance. Hybrid module for now contains only MeLiF algorithm that was developed at ITMO University [13].

As prerequisites for library usage is only numpy package is needed, the library is developed with Python 3 language. Current version of the library contents are shown below:

- Filters – folder which contains filters
  - Filter – class for constructing custom filter
  - MRMR – class for specific filter
  - VDM – class for specific filter
- Hybrid
  - MeLiF – class for basic Melif
- Wrappers
  - AddDel – class for ADD-DEL wrapper
  - Backward elimination – class for backward elimination wrapper
  - Sequential forward selection – class for sequential forward selection

Spearman correlation, Pearson correlation, information gain, Gini index, F-ratio, fit criterion, symmetric uncertainty are custom measures which are stored with cutting rules in Filter file. In the future when we will be adding more basic measures to the library and some more exotic cutting rules they could be easily added to the library through this file. MeLiF is one of a hybrid methods and it aggregates result of filters measures and wraps an estimator greedy optimising their ensemble weights. Pipeline displayed below on the Fig. 1:

As could be seen in Table I the Arizona State University feature selection library has the biggest number of implemented algorithms in comparison with others, but last time this library was updated about 2 years ago. Moreover, it doesn't have some basic feature selection algorithms. Scikit-learn feature selection module seems to have some rarely used methods and the general volume of presented algorithms is rather small. Feature extraction and selection book contains some examples of exotic methods; nevertheless, there are many different feature extraction methods there. Thus, though it is

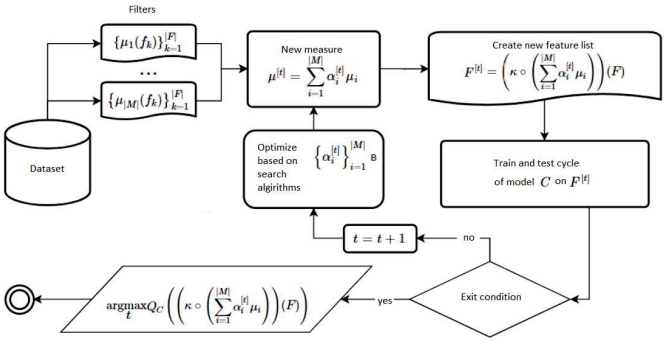

Fig. 1. Hybrid algorithm MeLiF pipeline

quite useful for dimensionality reduction in general, it is not usable in practice for feature selection. Development of ITMO Feature Selection library was started in the spring of 2019 and at this stage, the library is already implementing most of all basic architectural elements; it is sklearn compatible and user-friendly. It contains some basic algorithms used in practice as well as some less common methods.

V. ITMO_FS LIBRARY USAGE EXAMPLES AND PERFORMANCE TESTS

For all code samples following imports are required:

```
01 import time
02 from skfeature.function.
   statistical_based import gini_index
03 from sklearn.datasets import load_iris
04 from sklearn.feature_selection import
   SelectKBest
05 from sklearn.linear_model import
   LogisticRegression
06 from sklearn.metrics import f1_score
07 from sklearn.svm import SVC
08 from filters.Filter import *
09 from hybrid.Melif import Melif
10 from wrappers.AddDelWrapper import *
11 from wrappers.BackwardSelection
   import *
```

In the test_filter function example of the Filter class with Spearman correlation and “Best by value” cutting rule that equals to 0.99 is shown. This notation means that all features which have score higher than 0.99 will be selected. For this example iris [20] dataset was chosen.

```
01 def test_filter(self):
02   filtering = Filter("SpearmanCorr",
     GLOB_CR["Best by value"](0.99))
03   data, target = load_iris(True)
04   res = filtering.run(data, target)
05   print(data.shape, '--->', res.shape)
```

In the test_pearson_mat function example of Filter class with Pearson correlation and “Best by value” cutting rule that equals to 0.0 is shown. This example use orlraws10P dataset.

TABLE I. COMPARISON OF DIFFERENT PYTHON FS LIBRARIES CONTENTS

Algorithm	ITMO	ASU	SKL	FES
Correlation				
InformationGain				
GiniIndex				
F-ratio				
fit criterion				
MRMR				
VDM				
SymmUncertainity				
AddDel				
MeLiF				
CIFE				
CMIM				
DISR				
FCBF				
ICAP				
JMI				
MIFS				
MIM				
FisherScore				
ReliefF				
TraceRatio				
LL_L21				
LS_L21				
RFS				
ChiSquare				
T_score				
AlphaInvesting				
GraphFS				
GroupFS				
TreeFS				
DecisionTreeBackward				
DecisionTreeForward				
SVMbackward				
SVMforward				
LapScore				
SPEC				
MCFS				
NDFS				
UDFS				
ANOVAbased				
RFE				
FPR				
FDR				
FWE				
ROCfilter				
SimulatedAnnealing				

```
01 def test_pearson_mat(self):
02   data, target = self.orl['X'],
     self.orl['Y']
03   filtering = Filter("PearsonCorr",
     GLOB_CR["Best by value"](0.0))
04   res = filtering.run(data, target)
05   print(data.shape, '--->', res.shape)
```

In the test_pearson_k_best function example of Filter class with Pearson correlation and “K best” cutting rule with 6 best features selection is shown at line 04, and it is compared with scikit-learn SelectKBest usage at line 07. Test dataset here is Basehock. As can be seen from this example ITMO_FS template is more user friendly.

```
01 def test_pearson_k_best(self):
02   data, target = self.basehock['X'],
     self.basehock['Y']
03   start_time = time.time()
04   res = Filter("PearsonCorr",
```

```
    GLOB_CR["K best"](6))
    .run(data, target)
05  print("ITMO_FS time ---
    %s seconds ---" % (time.time() -
    start_time))
06  start_time = time.time()
07  res = SelectKBest(
    GLOB_MEASURE["PearsonCorr"],k=6)
    .fit_transform(data, target)
08  print("SKLEARN time --- %s seconds
    ---" % (time.time() - start_time))
09  print(data.shape, '--->', res.shape)
```

In the test_gini_k_best function example of Filter class with Pearson correlation and "K best" cutting rule with 6 best features selection is shown at line 04, and it is compared with scikit-learn SelectKBest usage at line 07. Test dataset here is Basehock. As in the previous example ITMO_FS template is also more user friendly in this case.

```
01 def test_gini_k_best(self):
02   data, target = self.basehock['X'],
     self.basehock['Y']
03   start_time = time.time()
04   res = Filter("GiniIndex",
     GLOB_CR["K best"](6))
     .run(data, target)
05   print("ITMO_FS time --- %s seconds
     ---" % (time.time() - start_time))
06   start_time = time.time()
07   res = SelectKBest(
     GLOB_MEASURE["GiniIndex"],k=6)
     .fit_transform(data, target)
08   print("SKLEARN time --- %s seconds
     ---" % (time.time() - start_time))
09   print(data.shape, '--->', res.shape)
```

In the test_add_del function example of AddDel (line 04) wrapper method with basic logistic regression (line 03) as estimator is shown. For quality measure function F_1 score was chosen. Test dataset here is Basehock.

```
01 def test_add_del(self):
02   data, target = self.basehock['X'],
     self.basehock['Y']
03   lr = LogisticRegression()
04   wrapper = Add_del(lr, f1_score)
05   wrapper.run(data, target)
06   print(wrapper.best_score)
```

In test_backward_selection function example of backward selection wrapper (line 04) method with logistic regression (line 03) as estimator is shown. Test dataset here is Basehock.

```
01 def test_backward_selection(self):
02   data, target = self.basehock['X'],
     self.basehock['Y']
03   lr = LogisticRegression()
04   wrapper = BackwardSelection(lr, 100,
     GLOB_MEASURE["GiniIndex"])
05   wrapper.fit(data[:, :200], target)
06   print(wrapper.best_score)
07   wrapper.fit(data[:, :300], target)
08   print(wrapper.best_score)
```

The test_melif function displays example of melif class with Support vector classifier (line 08) as estimator. Chosen filters are Gini index (line 03), fratio (line 04) and information gain (line 05). F_1 score chosen as quality measure for MeLiF (line 06).

```
01 def test_melif(self):
02   data, target = self.basehock['X'],
     self.basehock['Y']
03   _filters = [Filter('GiniIndex',
     cutting_rule=GLOB_CR
     ["Best by value"](0.4)),
04   Filter(GLOB_MEASURE["FRatio"]
     (data.shape[1]),
     cutting_rule=GLOB_CR["Best by
     value"](0.6)),
05   Filter('InformationGain',
     cutting_rule=GLOB_CR["Best by
     value"](-0.4))]
06   melif = Melif(_filters, f1_score)
07   melif.fit(data, target)
08   estimator = SVC()
09   melif.run(GLOB_CR['K best'](50),
     estimator)
```

The test_arizona example shows code usage comparison between ASU Feature Selection library and ITMO University library. This example was used to estimate execution time of Gini index and F-score index for both libraries. Result of this comparison can be found in the Table III.

```
01 def test_arizona(self):
02   data, target = self.coil['X'],
     self.coil['Y']
03   start_time = time.time()
04   features = gini_index.gini_index(data,
     target)
05   print("ARIZONA time --- %s seconds
     ---" % (time.time() - start_time))
06   start_time = time.time()
07   features = GLOB_MEASURE["GiniIndex"]
     (data,target)
08   print("ITMO time --- %s seconds ---"
     % (time.time() - start_time))
09   start_time = time.time()
10   features = f_score.f_score(data,target)
11   print("ARIZONA time --- %s seconds
     ---"% (time.time() - start_time))
12   start_time = time.time()
13   features = GLOB_MEASURE["FRatio"]
     (data.shape[-1])(data, target)
14   print("ITMO time --- %s seconds ---"
     % (time.time() - start_time))
```

The Table II shows time comparison between ITMO Feature Selection library and scikit-learn feature selection module at selecting k best features by Gini index and Pearson correlation. Basehock dataset has 1993 samples and 4862 features, COIL20 dataset has 1440 samples and 1024 features and orlraws10P has 100 samples and 10304 features.

TABLE II. COMPARISON OF SKLEARN FEATURE SELECTION MODULE AND ITMO FS LIBRARY COMPUTATIONAL TIME IN SECONDS

Dataset	Library	Pearson	Gini index
Basehock	ITMO_FS	0.129	0.373
	SKLEARN	0.122	0.379
COIL20	ITMO_FS	0.020	0.044
	SKLEARN	0.025	0.057
orlraws10P	ITMO_FS	0.019	0.049
	SKLEARN	0.017	0.049

As could be seen from the Table II the ITMO FS library has approximately the same computational time for Pearson correlation coefficient and Gini index filters as in the scikit-learn feature selection module. As these filters are not implemented in scikit-learn, we have put our customized measures in it for this reason.

TABLE III. COMPARISON OF ASU FEATURE SELECTION LIBRARY AND ITMO FS LIBRARY COMPUTATIONAL TIME IN SECONDS

Dataset	Library	F-score index	Gini index
Basehock	ITMO	1.084	0.376
	ASU	0.116	1.507
COIL	ITMO	1.081	0.048
	ASU	0.048	253.257
ORL	ITMO	5.793	0.058
	ASU	0.028	97.622

As could be seen from the Table III the ITMO FS library has a little bigger computational time for F-score Index and a lot smaller for Gini Index than Arizona State University feature selection library. As ASU library does not provide support for cutting rules in traditional way we ran intrinsic measures of ITMO FS algorithms separately from cutting rules for proper comparison.

VI. CONCLUSION

Lack of many general-purpose machine learning libraries in Python still exists, especially in fields that are not closely related to neural networks. For this reason in this paper, we have proposed a new Python feature selection library ITMO FS. To do this we have provided a review of existing approaches to feature selection and have performed a survey of existing Python feature selection libraries. It is shown that all of them are scikit-learn compatible and implementing only several feature selection algorithms, except Arizona State University feature selection library, which is the biggest one available for Python. We provided complete comparison with all surveyed libraries and implemented architecture for ITMO FS library that provides better view to the traditional feature selection algorithms categorization, support of hybrids and ensembles and in general more user-friendly than other ones. Moreover, our tests have shown that ITMO FS library works faster than ASU feature selection library on some algorithms. Also we have provided code samples for better understanding of library usage. In the future we plan to implement all algorithms mentioned in the Table I, add more modern algorithms of feature selection and more classical ones and then move to implementation of different ensembling and hybrid algorithms. In perspective, we also will add meta-learning approaches for easier selection of feature selection algorithms and their tuning.

ACKNOWLEDGMENT

This work is financially supported by the Government of the Russian Federation, Grant 08-08.

REFERENCES

[1] Y. Saeys, I. Inza, P. Larraaga, “A review of feature selection techniques in bioinformatics.”, *bioinformatics*,vol.23, num.19, 2007, pp. 2507-2517.

[2] L. Wang, Y. Wang, Q. Chang, “Feature selection methods for big data bioinformatics: A survey from the search perspective.”, *Methods*,vol.111, Dec.2016, pp. 21-31.

[3] J. Li, X. Hu, J. Tang, H. Liu, “Unsupervised streaming feature selection in social media.”, *In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, Oct 2015, pp. 1041-1050.

[4] Z.M. Hira, D.F. Gillies, (2015). “A review of feature selection and feature extraction methods applied on microarray data.”, *Advances in bioinformatics*, Jun.2015.

[5] T. Hastie, R. Tibshirani, J. Friedman, J. Franklin, “The elements of statistical learning: data mining, inference and prediction.”, *The Mathematical Intelligencer*, 2005, pp. 83-85.

[6] V. Boln-Canedo, N. Snchez-Maroo, A. Alonso-Betanzos, “A review of feature selection methods on synthetic data.” , *Knowledge and information systems*,vol.111, num.3, Mar.2012, pp. 483-519.

[7] I. Guyon, A. Elisseeff,. “An introduction to variable and feature selection.”, *JMLR*, Mar 2003, pp. 1157-1182.

[8] S. Raschka, V. Mirjalili, “Python machine learning.”, Packt Publishing Ltd, 2017.

[9] N. Ketkar, “Deep Learning with Python”, Apress, 2017.

[10] ITMO University feature selection library: https://github.com/LastShekel/ITMO_FS

[11] Arizona State University feature selection library: https://github.com/jundongl/scikit-feature

[12] Scikit-learn feature selection module: https://scikit-learn.org/stable/modules/feature_selection.html#feature-selection

[13] I. Smetannikov, A. Filchenkov, “MeLiF: filter ensemble learning algorithm for gene selection.”, *Advanced Science Letters*, vol.22, num.10, 2016, pp. 2982-2986.

[14] V. Bolon-Canedo, N. Sanchez-Marono, A. Alonso-Betanzos, J.M. Benitez, F. Herrera, “A review of microarray datasets and applied feature selection methods.”, *Information Sciences*, vol.282, 2014, pp. 111-135.

[15] Y. Saeys, T. Abeel, Y. Van de Peer, “Robust feature selection using ensemble feature selection techniques.”, *In Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Sep. 2008, pp. 313-325.

[16] V. Bolon-Canedo, N. Sanchez-Marono, A. Alonso-Betanzos, “An ensemble of filters and classifiers for microarray data classification.”, *Pattern Recognition*, vol.45, num.1, 2012, pp. 531-539.

[17] J. Li, K. Cheng, S. Wang, F. Morstatter, R.P. Trevino, J. Tang, H. Liu “Feature selection: A data perspective”, *ACM Computing Surveys (CSUR)*, vol.50, num.6, 2018, p.94.

[18] Python implementations of the Boruta all relevant feature selection method: https://github.com/scikit-learn-contrib/boruta_py

[19] Code and Data Sets for Feature Engineering and Selection by Max Kuhn and Kjell Johnson (2019): https://github.com/topepo/FES

[20] Iris Data Set: https://archive.ics.uci.edu/ml/datasets/iris

[21] The 20 Newsgroups data set: http://qwone.com/simjason/20Newsgroups/

[22] Columbia University Image Library: http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html

[23] The Database of Faces: https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

[24] General features selection based on certain machine learning algorithm and evaluation methods: https://github.com/duxuhao/Feature-Selection

[25] Scikit-feature is an open-source feature selection repository in Python developed at Arizona State University: http://featureselection.asu.edu/index.php

[26] Scikit-learn-compatible Python implementation of ReBATE: https://github.com/EpistasisLab/scikit-rebate