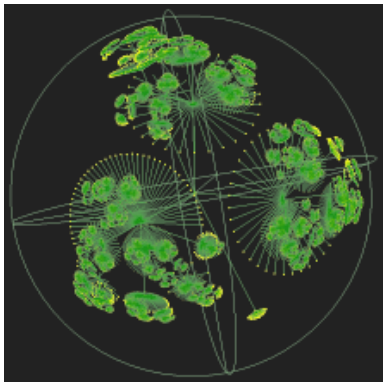


ESSeRE Lab

WekaNose Tutorial

Contents

1 What is it for?

WekaNose is an environment which allows to perform an experiment, that have as goal to studies the code smell detection through machine learning techniques. This experiment is divided in two main part: (i) the first one concern the creation of the dataset and (ii) the second part where the machine learning algorithms are trained and tested using the dataset created in the first part.

1.1 Dataset Creation

The first part can be separated in six steps, three of which can be performed using WekaNose.

- **Choice of the code smell and creation of a definition for it**
  The first thing to be done is select which code smell you want to take in consideration and create a definition for it, hopefully using some literature. This initial step is very important because the definition will affect the entire experiment, so make sure that the definition underline the code smell's characteristics on which you want to focus the most;

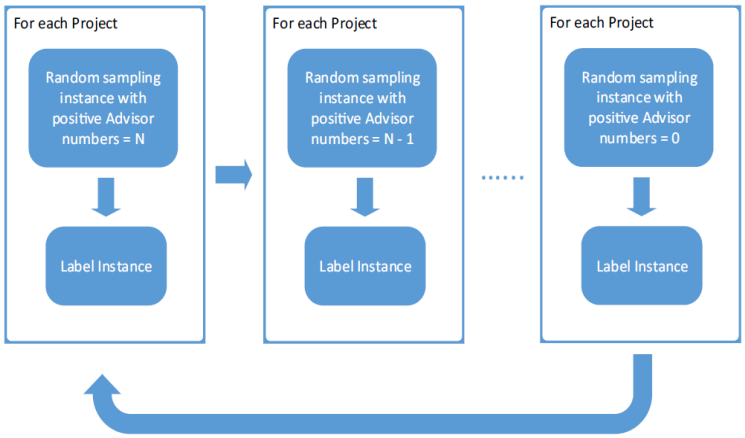
- **Collection of a large repository of heterogeneous software systems written in Java**
  The next step is to collect some software systems that will be analyzed and from which the instances of the dataset can be extracted. The systems have to be the most heterogeneous you can find (or create), so that the instances will not refers mostly to the some type of class (o method) that you can find in specific type of software system. An example of it is QualitasCorpus (`http://qualitascorpus.com/`);

- **Metrics Extraction**
  Then you have to extract a large set of object-oriented metrics from systems at class, method, package and project levels. The metrics are considered as independent variables in the machine learning approach. The selected metrics are at class, method, package levels. This step is performed by WekaNose using DFMC4J (Design Features and Metrics for Java) that is part of a bigger tool called JCodeOdor developed by the ESSeRe Lab, for more information about it `http://essere.disco.unimib.it/wiki/jcodeodor`.

- **Choice of the Advisors**
  An Advisor is a deterministic rule that gives a classification of a code element, telling if it is a code smell or not. The idea is that Advisors should approximate the label better than the random choice, and by aggregating their suggestions we should be able to sample more code elements affected by code smells. The rules can be extracted by some code smells detector or by some article, but, once again, the rules will affected the experiment so be sure that for each code smell, you select at least three Adivisor: (i)

one that most likely characterizes the not smell instances, (ii) one that most likely characterizes the affected instances, (iii) and one that include all the instances that can't be easily classified. WekaNose will apply the rules automatically. For more information about the metrics you can use to specify the advisors: `http://essere.disco.unimib.it/reverse/MLCSD.html`;

- **Sampling**
  Following the output of the Advisors, the instances will be divided in groups based on how many advisors consider the instances affected by the code smell. After that sampling process describe by figure below will be followed automatically by WekaNose;



- **Labeling**
  At this point the dataset is already created but the most important step has to be performed: for each instances contained in the dataset you have to manually evaluate it as affected or not affected by the code smell using the definition created at the first step. If the definition produced can't create a straight line that divide the instances affected by the code smell for the not affected ones, which most likely is going to happen, you can help yourself defining a serious of factor that characterize the code smell, they should be extracted by the definition itself if possible in order to minimize the aspect that can influence the result. Then order the factors from the one that, for you experimentation, is the most important to the less important one. Once you have you ordered list you have to keep it close to you for the hole evaluation process, and stick to that list every time you find an instance that for some reason is hard to evaluate

1.2 Machine Learning Approach

The second part can be separated in three steps, two of which can be performed using WekaNose. The hole process of training and testing of the machine learning algorithms is performed using OUTLINE (`https://github.com/UmbertoAzadi/OUTLINE`), that is basically a tool that allows to use the algorithms implemented in WEKA (`https://www.cs.waikato.ac.nz/ml/weka/`) through a more handy API.

- **Select the machine learning algorithms**
  This step require some knowledge about the machine learning algorithms, plus you may want to take in consideration some of the studies performed about the usage of machine learning techniques for code smells detection:

  - Arcelli Fontana, Francesca, Mntyl, Mika V., Zanoni, Marco, & Marino, Alessandro. 2016. Comparing and Experimenting Machine Learning Techniques for Code Smell Detection. Empirical Softw. Engg., 21(3), 1143 - 1191.
  - Fontana, F. A., Zanoni, M., Marino, A., & Mntyl, M. V. 2013 (Sept). Code Smell Detection: Towards a Machine Learning-Based Approach. Pages 396 - 399 of: 2013 IEEE International Conference on Software Maintenance.
  - Maiga, A., Ali, N., Bhattacharya, N., Sabane, A., Gueheneuc, Y., & Aimeur, E. 2012a. SMURF: A SVM-based Incremental Anti-pattern Detection Approach. Pages 466 - 475 of: Proceedings of the 19th Working Conference on Reverse Engineering (WCRE 2012). Kingston, Ontario, Canada: IEEE.
  - you can find a lot more just with a quick research on google

- **Apply the grid search**
  As you may know each machine learning algorithms has some parameters that it require in input. So the second step of the second part of this experiment is to find, for each algorithms selected, the best parameter. The parameters are considered the best if the performance of the machine learning algorithms are maximized by them, id est if there aren't any other values for the parameters that allows to reach a better performance. You can perform this step using WekaNose, in fact WEKA provide a serious of algorithms of its own to perform the greed search that can be used in WekaNose (`https://weka.wikispaces.com/Optimizing+parameters`)

- **Compare the machine learning algorithms with each others**
  At this point you should be quite sure about the machine learning algorithms to use and the how to initialize them in order to maximize the performance, so what is left to do is run the algorithms and choose the best one. You can perform this step using WekaNose, in fact one you configured all the algorithms and the dataset through WekaNose's

GUI you can Start the experiment, which means that a n-fold cross-validation tests with m repetitions will be performed for each classifiers, the result of this execution will be saved and compared using the corrected paired t-tests. (This procedure is already implemented in WEKA: `https://weka.wikispaces.com/Using+the+Experiment+API`)

2 How to install?

For install WekaNose you need:

- to download the repository from `https://github.com/UmbertoAzadi/WekaNose`
- to have installed **Java 8** or higher (for download https://www.java.com/it/download/)
- Right now this is not enough because some of the algorithms, for example LibSVM, create automatically a parameter -model ⟨WEKA path⟩, where WEKA path is C:\User\ProgramFiles\Weka in Windows, /home/user/Weka otherwise. Basically if you want to be sure that all the machine learning algorithms will run without problem you need to install WEKA:
  - Ubuntu: sudo apt-get install weka
  - Other: https://www.cs.waikato.ac.nz/ml/weka/downloading.html

3 How to run?

For run WekaNose you just need to

- open the command line:
  - Windows: WindowsKey+R and then type "cmd"
  - Ubuntu: Ctrl+Alt+t
- then type: "cd ⟨path of WekaNose⟩" (es: cd /home/user/Download/WekaNose)
- and finally type: java -jar WekaNose.jar

As you may come to understand on your own, this software were tested only with Windows (10 Pro) and Ubuntu (16.04 LTS)

N.B. Do NOT open the jar double-clicking on it, because the workspace's path (./WekaNose/result) will be compromized

4 Exemple of usage

In this section we will see how to perform all the steps presented above through WekaNose GUI. Take in consideration that at this point you should already select the code smell and the repository of software systems (first two point in the Section 1.1)

4.1 Dataset Creation

4.1.1 HomePage

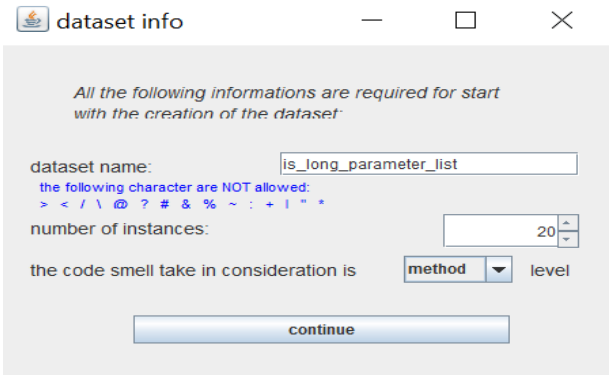
Assuming that this is a new study you have to press the button "Dataset Creation"

4.1.2 Dataset information

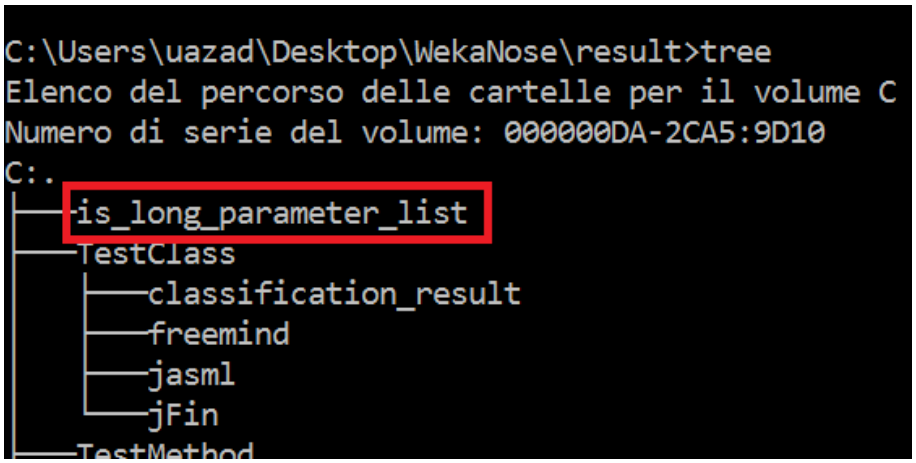
Here you must specify:

- the name of the dataset/experiment
- how many instances the dataset had to contain once it will be created
- if the code smell's level is: class or method

Then press continue.



As result of this first step you should see that a fold named as ⟨dataset name⟩ has been created in ./WekaNose/result



4.1.3 Load source

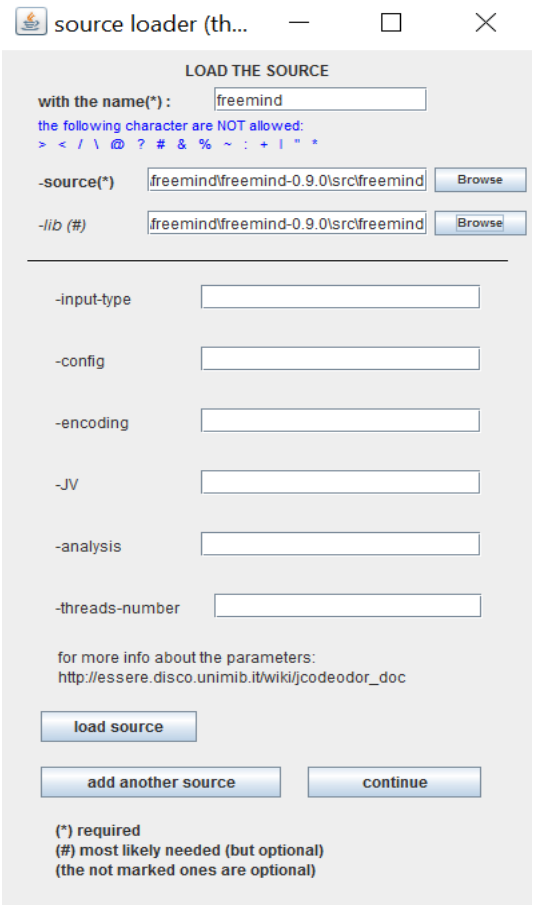
Now you have to specify one by one the source, id est the project you want to analyze. The parameter that you must specify are:

- the name of the source
- the path of the source

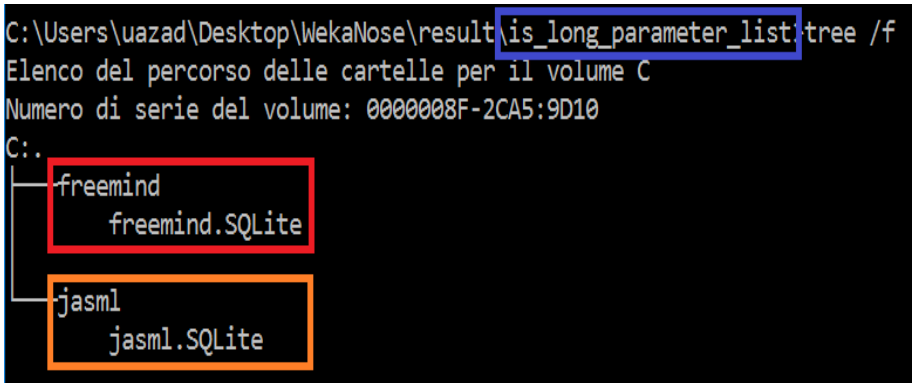
The optional ones are:

- the library needed by the source
- the remaining five can be ignored, unless you have some specific requirement, in that case you can find a more information here: http://essere.disco.unimib.it/wiki/jcodeodor_doc

One you specified all the information you have to press the button "load source", you should take in consideration that if the source is quite big it could take a few minutes. Then if you want to specify another source you have to press the button "add another source", otherwise you can press the button "continue".



As result of each loading you should see that, inside the fold created in the initial step, a fold named as ⟨name⟩ has been created and it contain a file called ⟨name⟩.SQLite, which contain the result of the analyzes

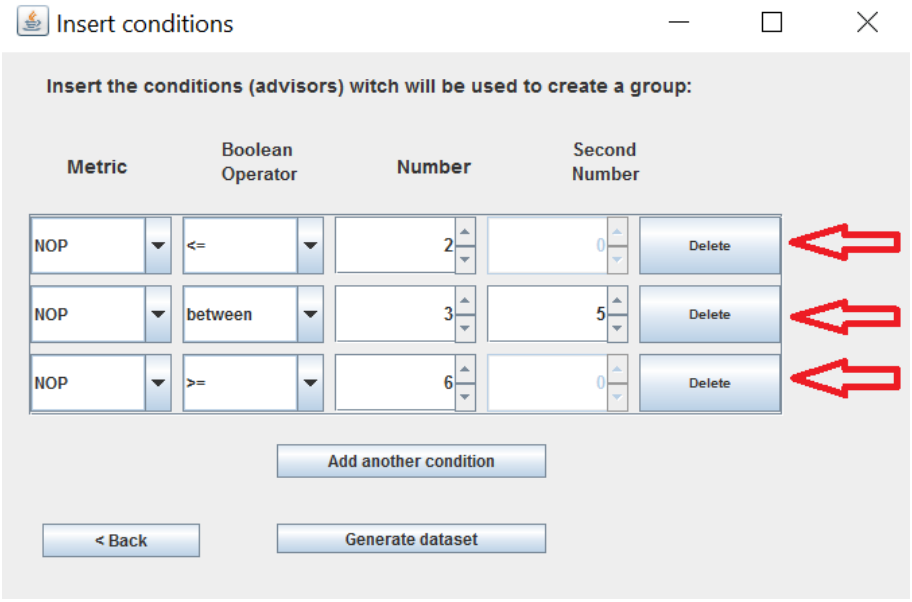


4.1.4 Specify the Advisors

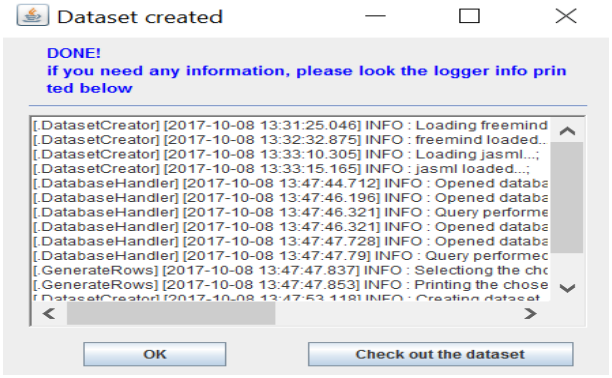
The next step is to specify the advisors, as you can see you can:

- "Add another condition"
- "Delete" an Advisor already created
- go "Back", in case you forgot to specify some sources

One you specified all the Advisors you have to press the button "Generate Dataset" (if you specified a lot of source you may need to wait a few minutes).

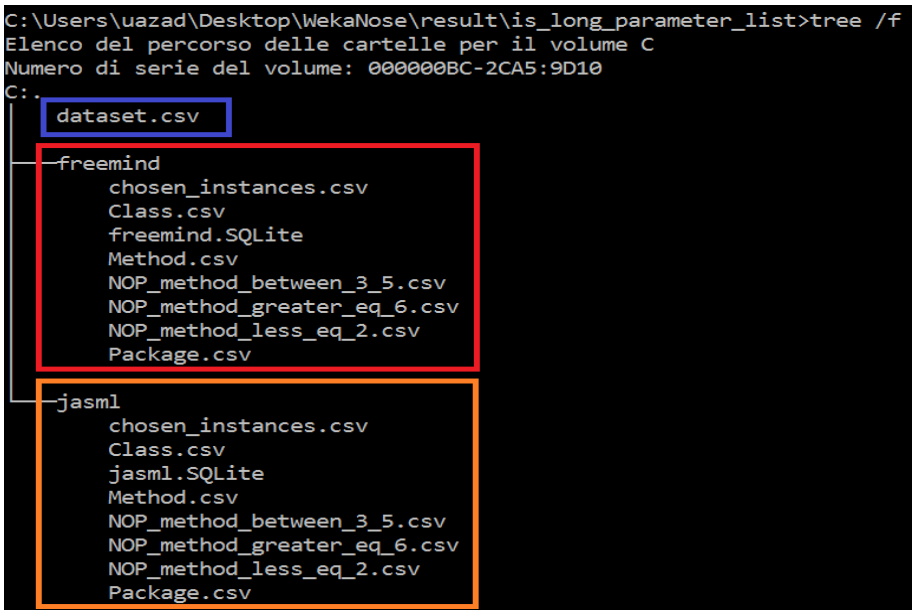


Once the dataset is generated the following windows will pop-up and you can open the directory where the dataset has been saved using the "Check out the result" button



As result of this step you should see that, inside the fold created in the initial step, a new file called dataset.csv has been created and inside of each source's fold some new files has been created:

- Method.csv contain all the metrics at method level
- Class.csv contain all the metrics at class level
- Package.csv contain all the metrics at package level
- One file for each advisors selected, witch will contain the instances (methods in this case) that match with the characteristic specified
- chosen_instaces.csv contain all the source's instances that have been chosen to be part of the dataset



4.1.5 Instances Labeling

If you open the file called dataset.csv it should be looking something like this:

	A	B	C	D	E	F	G	.....	BB	BC	BD
1	ID	Project	Package	Class	Method	NOP_method	CC_method		LOC_package	NOM_package	is_long_parameter_list
2	6401	freemind	freemind.modes.min	ImportAttributesDialog	getTreeCellRender	7.0	0.0		2362.0	136.0	
3	7192	freemind	plugins.collaboratior	SpringUtilities	makeGrid(javax.sv	7.0	0.0		730.0	33.0	
4	1925	freemind	freemind.modes.min	MindMapHTMLWriter	saveHTML(java.lar	6.0	0.0		4223.0	240.0	
5	4347	freemind	accessories.plugins.u	ClickableImageCreator	createArea()	0.0	0.0		101.0	4.0	
6	6124	freemind	plugins.collaboratior	MapSharingController.	mouseReleased(ja	1.0	0.0		884.0	23.0	
7	3666	freemind	accessories.plugins	NodeNote	startupMapHook(j	0.0	0.0		3705.0	164.0	
8	3019	freemind	freemind.view.mindr	ExtendedAttributeTabl	getRowCount()	0.0	0.0		1212.0	120.0	
9	7266	jasml	com.jasml.classes	Attribute_Code	Attribute_Code(in	9.0	1.0		1690.0	42.0	
10	7238	jasml	com.jasml.classes	Attribute	Attribute(byte[],in	3.0	1.0	.....	1690.0	42.0	
11	7339	jasml	com.jasml.compiler	ConstantPoolGenerato	addInterfaceMeth	4.0	1.0		2639.0	106.0	
12	7376	jasml	com.jasml.compiler	SourceCodeParser	getOffset(boolean	3.0	0.0		2639.0	106.0	
13	7285	jasml	com.jasml.classes	Attribute_InnerClasses	Attribute_InnerCl	3.0	2.0		1690.0	42.0	
14	1070	freemind	freemind.modes.min	ImportFolderStructure	addNode(freemin	3.0	0.0		5031.0	435.0	
15	3173	freemind	freemind.extensions	ImportWizard	addClassesFromDi	4.0	0.0		1339.0	136.0	
16	996	freemind	freemind.modes.min	MindMapActions	moveNodes(freem	3.0	0.0		5031.0	435.0	
17	1101	freemind	freemind.modes.min	PasteAction	_paste(DataFlavor	4.0	0.0		5031.0	435.0	
18	7247	jasml	com.jasml.classes	Constant_Long	getValue()	0.0	0.0		1690.0	42.0	
19	7407	jasml	com.jasml.compiler	Scanner	mark()	0.0	1.0		2639.0	106.0	
20	7416	jasml	com.jasml.compiler	Scanner	tokenType()	0.0	1.0		2639.0	106.0	
21	7447	jasml	com.jasml.helper	Util	accessFlagToString	1.0	1.0		943.0	34.0	

Now you have to label the instances, which means that you have to write in the last column “true” if you think that the instance take in consideration is affected by the code smell, “false” otherwise. Of course you don’t have to use “true” or “false”, any kind of label will do just as well.

	A	B	C	D	E	F	G	.....	BB	BC	BD
1	ID	Project	Package	Class	Method	NOP_method	CC_method		LOC_package	NOM_package	is_long_parameter_list
2	6401	freemind	freemind.modes.min	ImportAttributesDialog	getTreeCellRender	7.0	0.0		2362.0	136.0	true
3	7192	freemind	plugins.collaboratior	SpringUtilities	makeGrid(javax.sv	7.0	0.0		730.0	33.0	true
4	1925	freemind	freemind.modes.min	MindMapHTMLWriter	saveHTML(java.lar	6.0	0.0		4223.0	240.0	true
5	4347	freemind	accessories.plugins.u	ClickableImageCreator	createArea()	0.0	0.0		101.0	4.0	false
6	6124	freemind	plugins.collaboratior	MapSharingController.	mouseReleased(ja	1.0	0.0		884.0	23.0	false
7	3666	freemind	accessories.plugins	NodeNote	startupMapHook(j	0.0	0.0		3705.0	164.0	false
8	3019	freemind	freemind.view.mindr	ExtendedAttributeTabl	getRowCount()	0.0	0.0		1212.0	120.0	false
9	7266	jasml	com.jasml.classes	Attribute_Code	Attribute_Code(in	9.0	1.0		1690.0	42.0	true
10	7238	jasml	com.jasml.classes	Attribute	Attribute(byte[],in	3.0	1.0	.....	1690.0	42.0	false
11	7339	jasml	com.jasml.compiler	ConstantPoolGenerato	addInterfaceMeth	4.0	1.0		2639.0	106.0	true
12	7376	jasml	com.jasml.compiler	SourceCodeParser	getOffset(boolean	3.0	0.0		2639.0	106.0	false
13	7285	jasml	com.jasml.classes	Attribute_InnerClasses	Attribute_InnerCl	3.0	2.0		1690.0	42.0	false
14	1070	freemind	freemind.modes.min	ImportFolderStructure	addNode(freemin	3.0	0.0		5031.0	435.0	false
15	3173	freemind	freemind.extensions	ImportWizard	addClassesFromDi	4.0	0.0		1339.0	136.0	true
16	996	freemind	freemind.modes.min	MindMapActions	moveNodes(freem	3.0	0.0		5031.0	435.0	false
17	1101	freemind	freemind.modes.min	PasteAction	_paste(DataFlavor	4.0	0.0		5031.0	435.0	true
18	7247	jasml	com.jasml.classes	Constant_Long	getValue()	0.0	0.0		1690.0	42.0	false
19	7407	jasml	com.jasml.compiler	Scanner	mark()	0.0	1.0		2639.0	106.0	false
20	7416	jasml	com.jasml.compiler	Scanner	tokenType()	0.0	1.0		2639.0	106.0	false
21	7447	jasml	com.jasml.helper	Util	accessFlagToString	1.0	1.0		943.0	34.0	false

4.2 Machine Learning Tecnique

As you can see the windows is divided in three part:

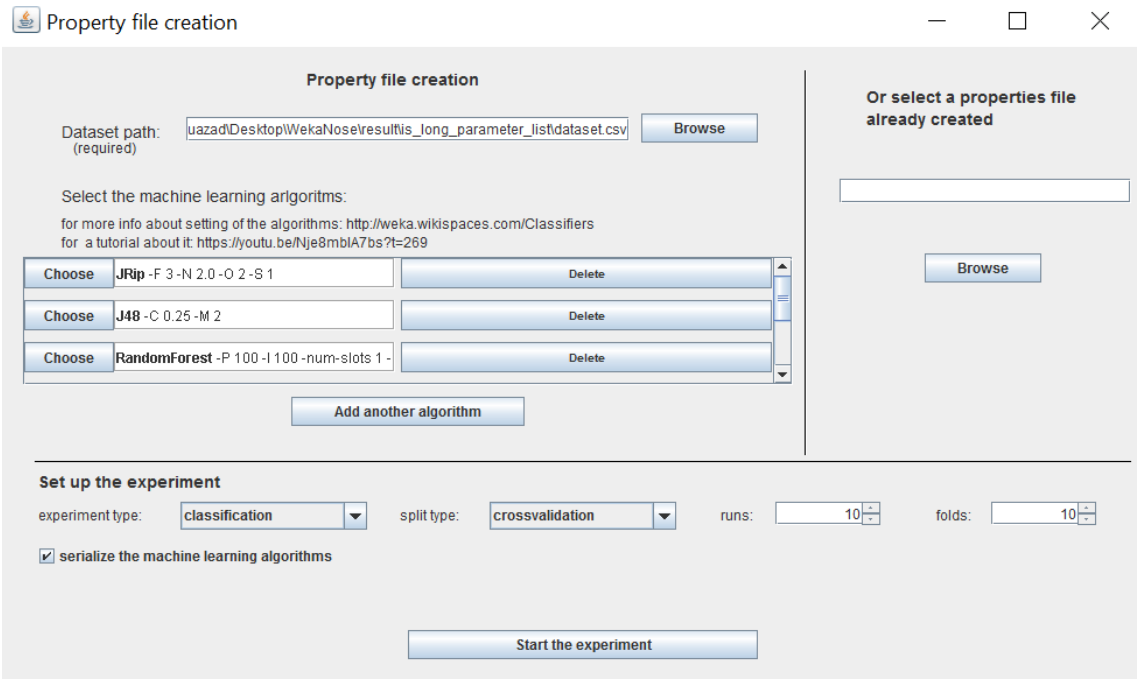
- the first part (top-left) allows to create a properties file by specifying the dataset path and at least one classifier. The properties file will be use by OUTLINE to perform the experiment.

- If you had already created the properties file, in a former experiment for example, you can select it using the the second part of the window (top-right), if you do that make sure that the "path" field of the properties file is consistent.
- finally the bottom part of the windows allows you to specify some experiment's characteristic:
  - the experiment type: classification (default), regression or custom
  - the experiment split type: cross validation(default), random split or custom

  For more information about the "custom" experiment: `https://github.com/UmbertoAzadi/OUTLINE`

  - how many times the experiment has to be repeated (default = 10)
  - in how many subset the dataset has to be split during the cross-validation (default = 10)

Once you insert all the information require you have to press the "Start the experiment" button, now based on which and how many many classifiers you selected the time required could be very high. For example during my stage i used a properties file with 32 classifier, 16 of which used the boosting technique AdaBoostM1, and it took 20 hour more or less.



As result of this step you should see that, inside the fold specified in the properties file's field called "path", a fold called classification_result is created, it will contain the following file:

- "Experiment_result.txt" that contain the ranked algorithms
- "ExperimentStatistics.arff" that contain the result of each classification/regression
- One file for each algorithm selected with extension ".txt" that contain the human-readable result of the classification. Take in consideration that not all the algoritms' result can be described in a human readable way.
- (Only if the flag "serialize" was selected) One file for each algorithm selected with extension ".model" that are the serialized algorithms