1. Background

1.1. Feature Selection

Feature Selection (FS) is defined in [**?**] as the process of eliminating features from the data base that are irrelevant to the task to be performed. FS facilitates data understanding, reduces the measurement and storage requirements, reduces the computational process time, and reduces the size of a data set, so that model learning becomes an easier process. An FS method is basically a *search strategy* where the performance of candidate subsets is measured with a given *evaluator*. The search space for candidate subsets has cardinality $O(2^w)$, where w is the number of features. A *stopping criterion* establishes when the feature selection process must finish. It can be defined as a control procedure that ensures that no further addition or deletion of features does produce a better subset, or it can be as simple as a counter of iterations. FS methods are typically categorized into *wrapper*, *filter* and *embedded*, *univariate* and *multivariate* methods. *Wrapper methods* [**?**] use a predetermined learning algorithm to determine the quality of selected features according to an evaluation metric [**?**]. *Filter methods* apply statistical measures to evaluate the set of attributes [**? ? ?**]. *Embedded methods* achieve model fitting and feature selection simultaneously [**?**]. *Multivariate methods* evaluate features in batches. *Univariate methods* evaluate each feature independently.

1.2. Feature Selection in Weka

In the *Waikato Environment for Knowledge Analysis* (*Weka*) [**?**], FS is implemented with the *weka.attributeSelection.AttributeSelection* class of the *weka.attributeSelection* package through two components: the *search strategy* (*weka.attributeSelection.ASSearch* abstract class) and the *evaluator* (*weka.attributeSelection.ASEvaluation* abstract class). This allows users and programmers to configure a multitude of different methods for FS, both filter and wrapper, univariate and multivariate. Evaluators with names ending in *SubsetEval* configure multivariate methods, whereas those with names ending in *AttributeEval* configure univariate methods. For multivariate wrapper FS methods, the *weka.attributeSelection* package has the *weka.attributeSelection.WrapperSubsetEval* class which evaluates attribute sets by using a learning scheme with cross-validation and a performance measure. For univariate wrapper FS methods, the *weka.attributeSelection.ClassifierAttributeEval* class evaluates the worth of an attribute by using a user-specified classifier, cross-validation and a performance evaluation measure to use for selecting attributes. Since the FS and classification processes must be executed in bach mode, Weka offers the class *weka.classifiers.meta.AttributeSelectedClassifier* which is a meta-classifier where dimensionality of data is reduced by attribute selection before being passed on to a learning algorithm.

2. Experiments and results

We have followed the methodology shown in the Figure 1 to perform FS. We have systematically applied 10 different FS methods for regression shown in Table 1 and graphically in Figure 2. In Table 1, *Database #Id* denotes the identifier of the reduced database generated with each FS method. Each FS method is the result of a specific choice among the search strategy, the evaluator, and the performance metric (for wrapper FS methods). We considered for this research seven wrapper FS methods and three filter FS methods. Among them, seven FS methods are multivariate and three FS methods are univariate. Table 2 shows the parameters used for each FS method.

Database #Id.	***FS method***	***Name***	***Search strategy***	***Evaluator***
#1	Wrapper Multivariate	*MOES-RF-RMSE*	*MultiObjectiveEvolutionarySearch*	*RandonForest (RMSE)*
#2	Wrapper Multivariate	*PSO-RF-RMSE*	*PSOSearch*	*RandonForest (RMSE)*
#3	Wrapper Multivariate	*MOES-RF-MAE*	*MultiObjectiveEvolutionarySearch*	*RandonForest (MAE)*
#4	Wrapper Multivariate	*PSO-RF-MAE*	*PSOSearch*	*RandonForest (MAE)*
#5	Wrapper Multivariate	*MOES-IBk-RMSE*	*MultiObjectiveEvolutionarySearch*	*IBk (RMSE)*
#6	Wrapper Multivariate	*PSO-LR-RMSE*	*PSOSearch*	*LinearRegression (RMSE)*
#7	Wrapper Univariate	*RANKER-RF-RMSE*	*Ranker*	*RandonForest (RMSE)*
#8	Filter Multivariate	*GS-CSE*	*GreedyStepwise*	*ConsistencySubsetEval*
#9	Filter Univariate	*RANKER-RFAE*	*Ranker*	*ReliefFAttributeEval*
#10	Filter Univariate	*RANKER-PCA*	*Ranker*	*PrincipalComponents*

Table 1: Proposed Feature Selection methods for regression.

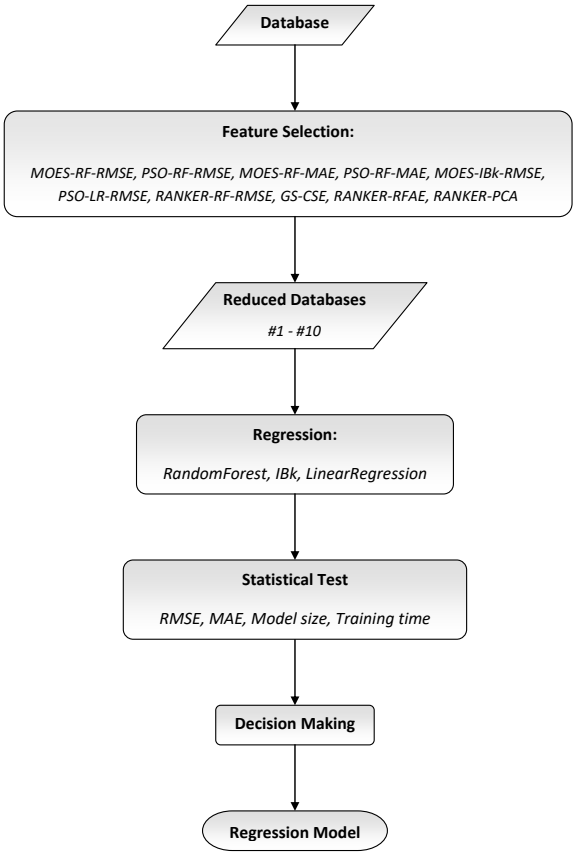

Figure 1: Metodology for Feature Selection for regression.

The following search strategies, evaluators, regression methods and performance metrics have been considered:

Search strategies For multivariate FS methods, among *deterministic search strategies* we considered *GreedyStepwise* [**?**], while the employed *probabilistic algorithms* are *MultiObjectiveEvolutionarySearch* [**?**] and *PSOSearch* [**?**]. *GreedyStepwise* performs a greedy forward or backward search through the space of attribute subsets, stopping when the addition (forward direction) or deletion (backward direction) of any of the remaining attributes results in a decrease in evaluation, thus, it has no backtracking capability. *MultiObjectiveEvolutionarySearch* use evolutionary computation where two objectives are optimized: the first one is chosen by the evaluator, and it is to be maximized, while the second one is the attribute subset cardinality, and it is to be minimized. The final output is given by the non-dominated solutions in the last population having the best fitness score for the first objective. *PSO* optimizes a problem moving individuals (particles) around in the search-space according to the particle's position and velocity, influenced by its local best known position and the best known positions.

For univariate FS methods, *Ranker* method [**?**] is required. *Ranker* method ranks attributes by their individual evaluations. A threshold, or the number of attributes to retain, allows reducing the attribute set.

Database #Id.	*Parameters*
#1	-E “weka.attributeSelection.WrapperSubsetEval -B weka.classifiers.trees.RandomForest -F 5 -T 0.01 -R 1 -E DEFAULT – -P 100 -I 10 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1” -S “weka.attributeSelection.MultiObjectiveEvolutionarySearch -generations 1000 -population-size 100 -seed 1 -a 0 ”
#2	-E “weka.attributeSelection.WrapperSubsetEval -B weka.classifiers.trees.RandomForest -F 5 -T 0.01 -R 1 -E DEFAULT – -P 100 -I 10 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1” -S “weka.attributeSelection.PSOSearch -N 20 -I 1000 -T 0 -M 0.01 -A 0.33 -B 0.33 -C 0.34 -S 1”
#3	-E “weka.attributeSelection.WrapperSubsetEval -B weka.classifiers.trees.RandomForest -F 5 -T 0.01 -R 1 -E MAE – -P 100 -I 10 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1” -S “weka.attributeSelection.MultiObjectiveEvolutionarySearch -generations 1000 -population-size 100 -seed 1 -a 0 ”
#4	-E “weka.attributeSelection.WrapperSubsetEval -B weka.classifiers.trees.RandomForest -F 5 -T 0.01 -R 1 -E MAE – -P 100 -I 10 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1” -S “weka.attributeSelection.PSOSearch -N 20 -I 1000 -T 0 -M 0.01 -A 0.33 -B 0.33 -C 0.34 -S 1”
#5	-E “weka.attributeSelection.WrapperSubsetEval -B weka.classifiers.lazy.IBk -F 5 -T 0.01 -R 1 -E DEFAULT – -K 1 -W 0 -A “weka.core.neighboursearch.LinearNNSearch -A “weka.core.EuclideanDistance -R first-last”” -S “weka.attributeSelection.MultiObjectiveEvolutionarySearch -generations 10 -population-size 100 -seed 1 -a 0”
#6	-E “weka.attributeSelection.WrapperSubsetEval -B weka.classifiers.functions.LinearRegression -F 5 -T 0.01 -R 1 -E DEFAULT – -S 0 -R 1.0E-8 -num-decimal-places 4” -S ”weka.attributeSelection.PSOSearch -N 100 -I 1000 -T 0 -M 0.01 -A 0.33 -B 0.33 -C 0.34 -R 1000 -S 1”
#7	-E “weka.attributeSelection.WrapperSubsetEval -B weka.classifiers.trees.RandomForest -F 5 -T 0.01 -R 1 -E DEFAULT – -P 100 -I 10 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1” -S “weka.attributeSelection.Ranker -T -1.8E308 -N 10”
#8	-E “weka.attributeSelection.CfsSubsetEval -P 1 -E 1” -S “weka.attributeSelection.GreedyStepwise -T -1.8E308 -N -1 -num-slots 1”
#9	-E “weka.attributeSelection.ReliefFAttributeEval -M -1 -D 1 -K 10” -S ”weka.attributeSelection.Ranker -T -1.8E308 -N 10”
#10	-E “weka.attributeSelection.PrincipalComponents -R 0.95 -A 5” -S “weka.attributeSelection.Ranker -T -1.8E308 -N 10”

Table 2: Parameters of the Feature Selection methods for regression.

Evaluators We considered the multivariate filter evaluator *ConsistencySubsetEval* [**?**]. *ConsistencySubsetEval* scores a subset of features as a whole, by projecting the training instances according to the attribute subset, and considering the consistency of class values in the obtained instance sets. As far as univariate filter evaluators are concerned, *ReliefftAttributeEval* [**?**] and *PrincipalComponents* [**?**] were considered. *ReliefftAttributeEval* evaluates the worth of an attribute by repeatedly sampling an instance and considering the value of the given attribute for the nearest instance of the same and different class. Can operate on both discrete and continuous class data. *PrincipalComponents* performs a principal components analysis and transformation of the data. Dimensionality reduction is accomplished by choosing enough eigenvectors to account for some percentage of the variance in the original data (default 95%). Attribute noise can be filtered by transforming to the principal components space, eliminating some of the worst eigenvectors, and then transforming back to the original space.

We use the wrapper *WrapperSubsetEval* [**?**] for multivariate FS methods and *ClassifierAttributeEval* [**?**] for univariate FS methods in conjunction with the predictors *RandomForest* [**?**], *IBk* [**?**] and *LinearRegression* [**?**], and with the metrics *root mean squared error* (*RMSE*) and *mean absolute error* (*MAE*) [**?**]. *RandomForest* is an

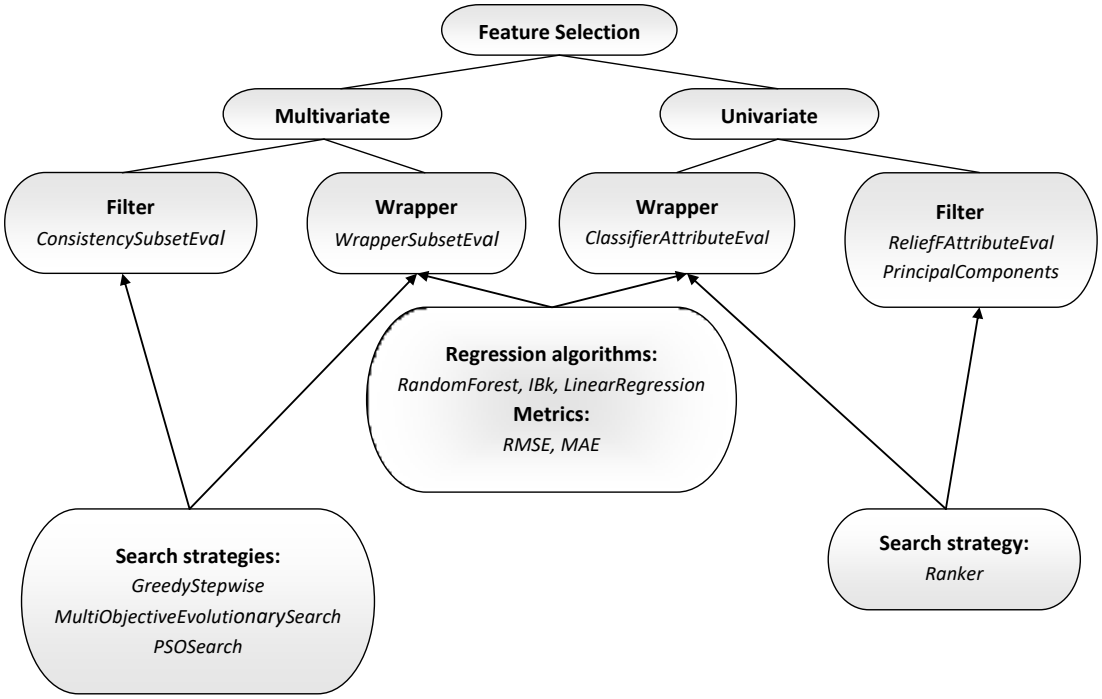

Figure 2: Organization chart of the proposed Feature Selection methods for regression.

ensemble learning method which constructs a forest of random trees with controlled variance, for classification or regression purposes. *IBk* is a simple instance-based learner that uses the class of the nearest k training instances for the class of the test instances and it is also valid for regressión. *LinearRegression* uses the *Akaike* criterion for model selection, and is able to deal with weighted instances.

Once FS was made, the next step was to perform regression with the reduced and original databases using *RandomForest*, *IBk* and *LinearRegression*. The Table 3 shows the parameters used for the regression methods. In order to detect over-fitting and prediction ability, the regression models have been evaluated in both full training set and 10-fold cross-validation over 10 iterations. Tables 4 and 5 show the evaluation in full training set for the *RMSE* and *MAE* metrics respectively. The Tables 6, 7, 8 and 9 show the evaluation in 10-fold cross-validation (10 iterations) for the metrics *RMSE*, *MAE*, *Serialized_Model_Size* and *UserCPU_Time_testing*[1] respectively. The result of the experiment has been analysed through a *paired t-test (corrected)*, with 0.05 significance (being *#1#2* the test base in Tables 6 and 7, and *RandomForest* in Tables 8 and 9). For each result, a mark $*$ denotes that the result is statistically worse than the test base; similarly, a mark *v* denotes a statistically better result, and no mark denotes no statistically meaningful difference. Note that the reduced databases *#1* and *#2* are the same, as are the reduced databases *#3* and *#4*, so they appear together in all the tables.

[1]Intel (R) Core (TM) i5-4460 @ 3.20 GHz 3.20 GHz RAM 8.00 GB Operating Systems 64 bits, processor x64.

Name	*Parameters*
RandomForest	-P 100 -I 500 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1
IBk	weka.classifiers.lazy.IBk -K 1 -W 0
	-A "weka.core.neighboursearch.LinearNNSearch
	-A "weka.core.EuclideanDistance -R first-last""
LinearRegression	-S 0 -R 1.0E-8 -num-decimal-places 4

Table 3: Parameters of the regression methods.

Database #Id.	*RandomForest*	*IBk*	*LinearRegression*
#1#2	4.2267	0.0	66.8625
#3#4	4.2734	15.8069	66.8642
#5	4.6102	0.0	69.5321
#6	9.3867	0.0	54.7665
#7	5.8233	0.0	62.1653
#8	6.8531	0.0	56.583
#9	10.9122	7.0239	56.5407
#10	16.6868	0.0	58.8544
Original	10.0615	0.0	54.6545

Table 4: *RMSE* with full training set.

3. Analysis of results and discussion

Clearly the best results have been obtained with the FS methods *MOES-RF-RMSE* / *PSO-RF-RMSE* (*#1#2*) and *MOES-RF-MAE* / *PSO-RF-MAE* (*#3#4*), which show statistically significant differences with respect to the rest of the analysed FS methods for the *RMSE* and *MAE* performance metrics. The Table 10 show the selected attributes with these FS methods. All attributes of *#3#4* also belong to *#1#2*, and the performances of the databases are similar in both full training set and 10-fold cross-validation (10 iterations). We can then conclude that the best selection of attributes is the solution *#3#4*.

The following general statements can be derived from the results:

- Wrapper FS methods have shown better performance for *RMSE* and *MAE* metrics than filter FS methods.
- Multivariate FS methods have shown better performance for *RMSE* and *MAE* metrics than univariate FS methods.
- For wrapper FS methods, *RandomForest* has proven more effective than *IBk* and *LinearRegression* based evaluators.
- Run time of *RandomForest* is acceptable for wrapper FS methods setting the number of iterations to 10 (-I 10), and the method is not very sensitive to the variation of its parameters. However, *RandomForest* generates large size regression models.
- *IBk* is very prone to over-fitting.
- *LinearRegression* is very fast and not prone to over-fitting, but it has not been efficient for this problem.

References

Database #Id.	*RandomForest*	*IBk*	*LinearRegression*
#1#2	2.2503	0.0	51.4108
#3#4	2.2696	4.7964	51.3844
#5	2.438	0.0	53.8089
#6	5.8176	0.0	39.4351
#7	3.273	0.0	44.8639
#8	4.005	0.0	40.8774
#9	5.9238	1.5123	41.52
#10	10.892	0.0	41.2654
Original	6.2188	0.0	39.3108

Table 5: *MAE* with full training set.

	#1#2	*#3#4*	*#5*	*#6*	*#7*	*#8*	*#9*	*#10*	*Original*
RandomForest	11.00	11.02	12.23 *	26.04 *	16.13 *	18.82 *	24.02 *	46.02 *	27.59 *
IBk	21.17	40.04 *	20.82 v	37.06 *	34.47 *	30.24 *	39.08 *	46.79 *	36.95 *
LinearRegression	66.89	66.88	69.55 *	55.09 v	62.28 v	56.72 v	56.66 v	58.93 v	55.31 v

Table 6: *RMSE* with 10-fold cross-validation (10 iterations).

	#1#2	*#3#4*	*#5*	*#6*	*#7*	*#8*	*#9*	*#10*	*Original*
RandomForest	5.19	5.15	5.70 *	15.93 *	8.51 *	10.64 *	13.54 *	29.98 *	16.82 *
IBk	10.19	19.91 *	9.99 v	17.31 *	16.17 *	13.74 *	20.07 *	21.21 *	16.69 *
LinearRegression	51.46	51.44	53.87 *	39.71 v	44.97 v	40.99 v	41.62 v	41.35 v	39.81 v

Table 7: *MAE* with 10-fold cross-validation (10 iterations).

	RandomForest	*IBk*	*LinearRegression*
#1#2	77954766.98	513355.60 v	54118.52 v
#3#4	75704860.26	403181.80 v	53715.00 v
#5	79702492.83	319109.36 v	6339.56 v
#6	91655663.29	1321302.40 v	57163.96 v
#7	82474001.83	586730.40 v	54166.12 v
#8	83994700.42	759559.00 v	7235.44 v
#9	87917552.94	539231.84 v	6418.92 v
#10	103251437.55	539979.40 v	7176.08 v
Original	89395544.24	2128847.60 v	58977.72 v

Table 8: *Serialized_Model_Size* with 10-fold cross-validation (10 iterations).

	RandomForest	*IBk*	*LinearRegression*
#1#2	0.18	0.11 v	0.00 v
#3#4	0.19	0.09 v	0.00 v
#5	0.18	0.08 v	0.00 v
#6	0.26	0.23 v	0.00 v
#7	0.22	0.10 v	0.00 v
#8	0.23	0.13 v	0.00 v
#9	0.23	0.09 v	0.00 v
#10	0.23	0.10 v	0.00 v
Original	0.24	0.29 *	0.00 v

Table 9: *UserCPU_Time_testing* with 10-fold cross-validation (10 iterations).

Database #Id.	*Attributes*
#1#2	time, hour, stMO12_IMI_prec, stMU62_IMI_prec, month, day, dow, holiday
#3#4	time, hour, day, dow, holiday

Table 10: Best selected attributes.