Ensemble effort estimation using dynamic selection

Summary

SOFTWARE PRESENTATION

The software is partitioned in 5 packages of java classes:

- data
- main
- method
- model
- util

Below we will give a brief description about each package.

Package: data

The **data** package contains Java classes that represent databases. The package contains a public and abstract Java class named **"Padrao.java"**. This class contains abstract Java methods and constants used in the child classes. All concrete Java classes in the package must inherit the **"Padrao.java"** class. Each concrete Java class represents a real database that has been evaluated by the researchers.

Package: main

This package contains classes that execute the method evaluation code. All classes contain a main method that runs the program. The classes in this package can run one or more ML methods. The values assigned to the code variables define the configuration of the experiment.

Package: method

This package contains the classes that implement the methods evaluated in the experiments. From individual methods through static ensembles to the methods proposed in this work. The class **"RegressorIndividualGeral.java"** implements the individual methods, the individual model is passed by parameter to the class. Static ensembles are implemented in the **“EnsembleEstaticoGeral class”**, the base models of the ensemble are passed to the classes. Dynamic selection methods are implemented separately, in specific classes.

Package: model

This package contains classes that represent domain objects used in the experiments. For example, the representation of a software project, which is an instance of a database.

Pacote util

This package contains classes useful for the system. For example: classes with constant values and classes that define types of combination and validation methods.

RUNNING A SIMPLE INDIVIDUAL MODEL

Steps to run an individual machine learning method:

1. Create a Java class of type `Padrao`**.java** in the package **data**. The created class has to inherit the class `Padrao`**.java**. The created class should be similar to the concrete classes in the **data** package. The file containing the database must be in the .arff format used by the Weka Java code library, Note that the concrete classes will also configure a file in this format. We created 16 concrete classes of the type "Padrao.java" to be able running the experiment. Each class refers to a database.

2. In the **main** package, access and modify the class `TesteIndividualGeral`**.java** according to your needs. Then assign variable *`regressor`* an instance of any model. For example, to create a model ***`SUPPORT_VECTOR_REGRESSION`***, use:

   ```
   TesteIndividualGeral.regressor = new
   WekaExperiment().createClassifier(WekaExperiment.
   SUPPORT_VECTOR_REGRESSION);
   ```

   Other model examples can be found in the **util** package. `WekaExperiment`**.java**. The Java method `createClassifier(int theClassifier)` returns a classifier model (regressor/classifier) according to the integer value entered in the parameter.

3. In the class `TesteIndividualGeral`**.java**, you should set the values of the class constants `Constantes`**.java**.

   In this case it is needed to set: `Constantes`.*`BASE_VALIDACAO`* to inform to the experiment code if the evaluation will be based on the validation database (true) or the test database (false).

   ```
   Constantes.BASE_VALIDACAO = false;
   ```

4. The line of code initializes the lists that will save the accuracy results of the evaluated method for each metric.

   ```
   Utilidade.inicializaListasMetricas();
   ```

5. Then instantiate an object of the **method.**`RegressorIndividualGeral`**.java** class and pass the model created in the ***`regressor`*** variable to the class constructor.

   ```
   new RegressorIndividualGeral(regressor).run(new
   PadraoMiyazaki94(), TipoValidacao.LEAVE_ONE_OUT);
   ```

According to the example, this individual model will be executed in the database defined at Class `PadraoMiyazaki94`**.java** and ***`LEAVE_ONE_OUT`*** will be used as a validation method.

6. The rest of the code of the `TesteIndividualGeral`**.java** class will construct the output file name with the results.

Note that we can also evaluate the algorithms **Bagging**, **Boosting** e **Stacking** at `TesteIndividualGeral.java`. The *`regressor`* variable should be configured as follow for **bagging**, **boosting** and **stacking**, respectively.

```
        // Bagging
        Bagging bagging = new Bagging();

        bagging.setClassifier(new
        WekaExperiment().createClassifier(WekaExperiment.SUPPORT_VECTOR_
        REGRESSION));

        regressor = bagging;


        // Boosting
        AdditiveRegression additiveRegression = new
AdditiveRegression();

        additiveRegression.setClassifier(new
        WekaExperiment().createClassifier(WekaExperiment.SUPPORT_VECTOR_
        REGRESSION));

        regressor = additiveRegression;

        // Stacking

        Classifier regressor1 = new
        WekaExperiment().createClassifier(WekaExperiment.LEAST_MED_SQ);
        Classifier regressor2 = new
        WekaExperiment().createClassifier(WekaExperiment.SUPPORT_VECTOR_
        REGRESSION);
        Classifier regressor3 = new
        WekaExperiment().createClassifier(WekaExperiment.M5P);

        Stacking stacking = new Stacking();
        stacking.setMetaClassifier(new SMOreg());
        Classifier[] classifiers = new Classifier[3];
        classifiers[0] = regressor1;
        classifiers[1] = regressor2;
        classifiers[2] = regressor3;
        stacking.setClassifiers(classifiers);
```

RUNNING A STATIC ENSEMBLE FROM INDIVIDUAL MODELS

1. In the **main** package go to class `TesteEnsembleEstatico`. Set the value of the *experimento* variable; this variable is part of the identifier name of the output file generated with the results.

2. In the same way as in the previous procedure, define the value of the constant (true/false), and you should initialize the lists that will save the accuracy results.

```
Constantes.BASE_VALIDACAO = false;
Utilidade.inicializaListasMetricas();
```

3. Define the combination method used to merge the individual estimates for each regression model.

```
metodoDeCombinacao = TipoMetodoCombinacao.MEDIA;
```

   In the example we are using the mean as a combination method.

4. Instantiate the individual regression models that will be used in the ensemble.

```
regressor1 = new
WekaExperiment().createClassifier(WekaExperiment.LEAST_MED_SQ);
regressor2 = new
WekaExperiment().createClassifier(WekaExperiment.SUPPORT_VECTOR_REGRESSION);

regressor3 = new
WekaExperiment().createClassifier(WekaExperiment.M5P);
```

   In the example we are using the 3 regressors selected for the basic ensemble of the experiment presented in the work.

5. Instantiate a java object of the class `EnsembleEstaticoGeral`**.java** and pass the regressors created in the previous step as a parameter. Call the method `public void run(Padrao padrao, TipoMetodoCombinacao ensembleEstatico, TipoValidacao tipoValidacao)` of the class `EnsembleEstaticoGeral`**.java**.

```
new EnsembleEstaticoGeral(regressor1, regressor2, regressor3,
null, null).run(new PadraoMiyazaki94(), metodoDeCombinacao,
TipoValidacao.LEAVE_ONE_OUT);
```

   The `run` method receives the parameters: (i) an object of type `Padrao`**.java**, (ii) a combination method belonging to `TipoMetodoCombinacao` **enum**, and (iii) the type of validation used.

6. Call the method (`Utilidade.gerarArquivosMassa`) to generate the output file with the results.

```
Utilidade.gerarArquivosMassa(0, "ENSEMBLE_" + metodoDeCombinacao
+ experimento);
```

Running a dynamic selection method (based in distance)

1. Go to the `TesteKnoraDCSSetEnsembleDynamic` class in the **main** package. This class can run the DCS_LA, DCS_LAW, KNORA_U and KNORA_E methods

2. Similar to the previous steps, define the experiment name and the value of the constant that identifies the validation type, and initialize the lists to save the results.

```
experimento = "Myazaki94";
Constantes.BASE_VALIDACAO = false;
Utilidade.inicializaListasMetricas();
```

3. Inform the dynamic selection method that will be evaluated. In the example below we are using **Dynamic Classifier Selection by Local Accuracy (DCS_LA)**. It is also necessary to inform the evaluation metrics used for the algorithm to find the nearest neighbors. In our example, the absolute error was used as the metric to be considered. In the experiment presented in the paper, we repeated this assessment for the 3 metrics used.

```
metodoDeSelecaoPorDisntacia = TipoMetodoCombinacao.DCS_LA;
Utilidade.METRICA_AVALIACAO = Constantes.MAR;
Constantes.TIPO_METRICA_AVALIACAO = TipoMetricaAvaliacao.MAR;
```

4. Instantiate the regressors.

```
regressor1 = new
WekaExperiment().createClassifier(WekaExperiment.LEAST_MED_SQ);
regressor2 = new
WekaExperiment().createClassifier(WekaExperiment.SUPPORT_VECTOR_REGRESSION);
regressor3 = new WekaExperiment().createClassifier(WekaExperiment.M5P);
```

5. Create an object of the `Knora_DCS_Set_Ensemble_Dynamic` class and pass the 3 regressors previously created to the class's constructor. Then call the `run` method passing the requested parameters. The procedure is similar to the previous steps.

```
new Knora_DCS_Set_Ensemble_Dynamic(regressor1,regressor2,regressor3).run(new
PadraoMiyazaki94(), metodoDeSelecaoPorDisntacia,TipoValidacao.LEAVE_ONE_OUT);
```

6. Finally, call the `Utilidade.gerarArquivosMassa` method to create the file with the results obtained.

```
Utilidade.gerarArquivosMassa(0, "DYNAMIC_SELECTION_" + experimento +
metodoDeSelecaoPorDistancia);
```

RUNNING THE DYNAMIC SELECTION METHOD PROPOSED BY DI NUCCI

1. Access the `TesteExperimentoNucci` class in the **main** package. This class runs the method proposed by NUCCI.

2. Similar to the previous steps, define the name of the experiment and the value of the constant that identifies the validation type. Initialize the variables to store the metric.

```
String experimento = "Experimento_NUCCI_MAR";
String avaliacao = "Teste";

Constantes.BASE_VALIDACAO = false;
Utilidade.inicializaListasMetricas();
```

3. Define the metric that will be used to validate regressors in training. In the example we are using the absolute error.

```
Utilidade.METRICA_AVALIACAO = Constantes.MAR;
Constantes.TIPO_METRICA_AVALIACAO = TipoMetricaAvaliacao.MAR;
```

4. Define the number of classifiers to be used in the dynamic selection.

```
Constantes.QUANTIDADE_CLASSIFICADOR = 1;
```

5. Instantiate the regressors.

```
	Classifier regressor1 = new
WekaExperiment().createClassifier(WekaExperiment.LEAST_MED_SQ);
	Classifier regressor2 = new
WekaExperiment().createClassifier(WekaExperiment.SUPPORT_VECTOR_REGRES
SION);
	Classifier regressor3 = new
WekaExperiment().createClassifier(WekaExperiment.M5P);
```

6. Define the type of dynamic selection. We going to use simple dynamic selection in this example.

```
TipoMetodoCombinacao metodoDeCombinacao = TipoMetodoCombinacao.SD;
```

7. Instantiate an object of the `Set_Ensemble_Dynamic` class. Pass the 3 regressors in the class constructor. Call the `run` method and pass the requested parameters, similar to the previous steps. However, it is necessary to inform the value `WekaExperiment.RANDOM_FOREST` in the `Integer tipoClassificador7` parameter. It identifies the classifier used in the dynamic selection.

8. Call the method to generate the output file with the results.

```
	Utilidade.gerarArquivosMassa(0, "SET_DYNAMIC_SELECTION_RANDOM_FOREST"
+ experimento + "_" + metodoDeCombinacao + "_PEETACODS_" +  avaliacao);
```

RUNNING THE PROPOSED DYNAMIC ENSEMBLE SELECTION

1. Access the `TesteExperimentoPEETACO` class in the **main** package. This class runs the methods proposed in this work.

2. Similar to the previous steps, define the experiment name and the validation type constant. Define the metric that will be used to validate regressors in training.

```
Utilidade.METRICA_AVALIACAO = Constantes.MAR;

Constantes.BASE_VALIDACAO = false;
Constantes.TIPO_METRICA_AVALIACAO = TipoMetricaAvaliacao.MAR;

String experimento = "PEETACO";
String avaliacao = "Teste";
```

   In the example we are using the absolute error.

3. Instantiate the regressors.

```
Classifier regressor1 = new
WekaExperiment().createClassifier(WekaExperiment.LEAST_MED_SQ);
Classifier regressor2 = new
WekaExperiment().createClassifier(WekaExperiment.SUPPORT_VECTOR_REGRESSION);
Classifier regressor3 = new
WekaExperiment().createClassifier(WekaExperiment.M5P);
```

4. Define how many classifiers will be used in the dynamic selection (`Constantes.QUANTIDADE_CLASSIFICADOR`). If you want to use only 1 classifier just call the `runPeetacoDS` method.

```
runPeetacoDS(experimento, regressor1, regressor2, regressor3,
avaliacao);

Constantes.QUANTIDADE_CLASSIFICADOR = 3;
```

5. Call the `runPeetacoDES` dynamic ensemble selection method. This method creates the instance of the class responsible for making the selection.

6. Initialize the variables that store the result list.

```
Utilidade.inicializaListasMetricas();
```

7. Similar to the previous cases, pass the regressors as a parameter of the constructor of the class Set_Ensemble_Dynamic. Call the run method and inform the classifiers that will make the dynamic selection of the regressors. Classifiers must be chosen from a validation process carried out before the testing process. In the example, we use KNN with different k values to select the regressors.

```
            new Set_Ensemble_Dynamic(regressor1, regressor2,
regressor3).run(new PadraoMiyazaki94(), null, null, null, null,
WekaExperiment.KNN3, WekaExperiment.KNN7, WekaExperiment.KNN5,
metodoDeCombinacao, TipoValidacao.LEAVE_ONE_OUT);
```

8. Create the output files with the results

```
    Utilidade.gerarArquivosMassa(0, "SET_ENSEMBLE_DYNAMIC_" +
experimento + "_" + metodoDeCombinacao + "_PEETACODES_" +
Constantes.QUANTIDADE_CLASSIFICADOR + "CLASSIFICADORES" + avaliacao);}
```

9. Different classifiers can be used for different databases. In the example, the classifiers chosen were validated previously in the evaluated database.

10. The experiment can have different configurations. In this document, we present only one configuration for one of the databases used in the experiment presented in the article.

11. The `TesteExperimento` class can be configured to run multiple methods in a single run.