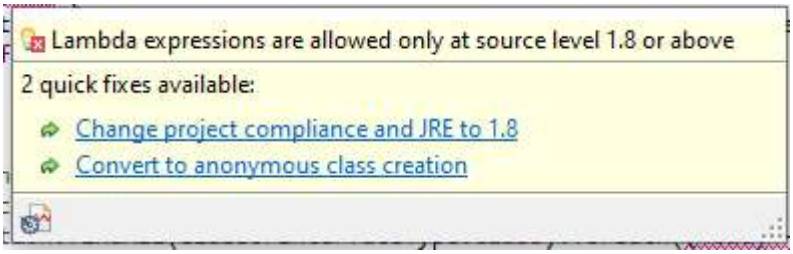
Guidelines

Table of Contents

Simple Summary

The most basic setup even requires some work to do. This is the list of things to do for the simplest experiment scenario, this scenario does not include the gui, it should be run from an IDE or console:

- First setup the codebase by using maven and installing the required libraries.
    - It is important that your project library must have its compliance and library as JRE1.8 or above, otherwise you will see errors like this:



- There are two options for datasets
    - First one is to import the database dump I have given in a database (h2 is preferred)
        - DROP TABLE IF EXISTS FEATURE_VECTOR;
        - RUNSCRIPT FROM 'feature_vector_dump';
        - SELECT COUNT(1) FROM FEATURE_VECTOR;
    - Set the database connection settings under Settings.java.
    - In order to use these features you need to uncomment the method block under "CloneClassification.java" lines 85-89
- Second option for dataset is using the .arff weka files I have given.
    - Place the .arff files anywhere you want
    - In "CloneClassification.java" file uncomment the lines 94~104, here replace the trainPath and testPath strings with the paths for the .arff files. Normally you will see that feature type and split is also important but, I cannot upload all different .arff files due to space limitations. So I have given an example .arff for both 10-features and all-features.
- After settings up either of the datasets run the very first method under App.java entry point which is:
    - `MultipleModelTraining mmt = new MultipleModelTraining();`
    - `mt.TrainAllModels();`
- It is a good idea to look in the method to see which classifier-type combinations will run because it might take some time to finish if default ones are left.

Description of each content uploaded in this directory

This directory includes five different utilities

First one is the code.rar which includes all files for a maven project, code is written in .java. There are multiple ways for running the code which I tried to describe in the code, however it might be hard to run directly since it requires a lot of path management.

Second directory is the "example datasets" where four .arff files and SQL DUMP can be found. These include equal splitted train and test set in .arff format. SQL DUMP can be imported to a database which is explained in the Simple Setup scenario.

Third directory is the "example trained models" which includes three fully trained models for J48, RandomForest and RandomCommittee.

Fourth file is the sql queries, which can be used in the BigBenchEval h2 database after processing the IJaDataset java files into the database. This processing part can be done using the .java code however you need to fix the path for the IJaDataset before using the corresponding method. It is also quite necessary to add unique indexes for the tables in the BigBenchEval database, otherwise queries will run slower than a turtle. It is not necessary to do this because I have given enough datasets for testing simple scenarios.

Fifth is the gui.rar which includes the graphical user interface designed in winforms. In order to use it, just import the project in visual studio then run it. However, under Form1.cs three paths needs to be modified accordingly to where the corresponding values are so that the program can find them.

- pathToJar: path to compiled .jar file
- pathToFindModel: path to find trained model - weka model
- pathToOutput: path to where the .jar file writes its prediction outputs

Setting up the code base

In order to run the model training and testing program, first extract the "code.rar" file. Since it is a maven java Project, I suggest importing it to Eclipse or any other IDE which supports the maven. However, it is also possible to use the entire system from command line as well. But I used Eclipse IDE. This will also help you to install the necessary libraries required to run the application.

Also, inside the codebase, I have gathered the necessary inputs required under Settings.java file.

How to run different experiments

There are four different main methods which can be called under App.java, which is the entry point for the application. Just comment the ones you don't need to use and uncomment the ones you will use. First two will be enough to just test general training and testing processes. Other two are additional utilities for processing data and command line arguments.

First one trains the models according to different combinations, models and feature types. It also prints the 10-Fold Cross Validation results, train-test data distributions, test evaluation results and the confusion matrices. Under App.java use the following section for this purpose. Inside "TrainAllModels" method just add-remove the required classifier types, feature types or split types to see which result you want.

```
MultipleModelTraining mmt = new MultipleModelTraining();
mmt.TrainAllModels();
```

Second one labels unlabelled data, as in it detects clones. Under App.java use the code given in the image. Inside "DetectClones", you will see that "ClassifyGivenProject" takes 5 parameters. These parameters can be set from Settings.java file. Settings.java explains what each variable is used for.

```
LabelUnlabelledData lud = new LabelUnlabelledData();
lud.DetectClones();
```

Third one is the process of reading the features from IJaDataset and writing them to database. Under Settings.java set "*IJaDataSetDirectory*" variable to where the location of the dataset is. You do not have to do this step. Because I have already provided necessary preprocessed datasets which can be used during training and testing. But in case you do, after running this step make sure to use the entire sql queries I have given with the project.

```
FeaturesToDatabase ftd = new FeaturesToDatabase();
ftd.ProcessData(false, true);
```

Fourth one is to run the application using a compiled .jar file. After compiling the project using either javac or any IDE. Users can trigger the prediction process using command line. There are three arguments "`pathToFindModel, pathToOutput, pathToFindClones`" which corresponds to, where the trained model is located, where will be the prediction output is gonna be written, and location where source codes are located. Just use the following code under App.java before compiling the code and comment everything else.

```
CommandLineTrigger clt = new CommandLineTrigger();
clt.CommandLineExecution(args);
```