SMODeLKit Tutorial

Joseph Heydorn

Contents

1 Introduction

SMODeLKit (Simple Multi-Output DependencE Learning toolKIT) is an open source toolkit for multi-dimensional classification, written in Java. The main advantage of SMODeLKit over existing machine learning toolkits is that it allows learned models to predict multiple output vectors, and it has methods for evaluating predictions with multiple output vectors.

1.1 Concepts

There are 3 basic concepts in SMODeLKit: learners, filters, and evaluators. Each of these concepts corresponds to a package with the same name.

- Learners - A learner is a machine learning model. It takes some vector of values as inputs, and predicts vectors of outputs.
- Filters - A filter is a way of transforming instances in a dataset. For example, the smodelkit.filter.Normalize scales real values to be between -1 and 1. Learners optionally have a filter which is applied to both training at test data before the learner trains or predicts. This way filters transform the data for a learner.
- Evaluators - An evaluator gives a score to predictions from a learner.

2 Getting Started

Prerequisite: Make sure you have installed Java JDK version 8 or later. Command line examples are given for a Linux bash shell, but similar commands should work in a windows command prompt.

To build SMODeLKit, set your current working directory to the project directory of SMODeLKit and run

```
ant build
```

To run SMODeLKit, run

```
java -cp 'bin:libraries/*' smodelkit.MLSystemsManager
```

You should see the usage printed with a list of command line options. For a simple example of how to train and evaluate a learner, run

```
java -cp 'bin:libraries/*' smodelkit.MLSystemsManager -L zeror model_settings/zeror.json \
 -A Datasets/mdc/synthetic/continuous_2out_4class.arff \
-E random 0.2
```

This runs a very simple classifier, ZeroR, on a synthetic dataset. ZeroR simply predicts the single most common output vector from the training set, ignoring the input. To see what each option means, see the usage statement printed in the previous step.

3 Dataset Format

SMODeLKit uses the arff format for datasets (for details see `http://www.cs.waikato.ac.nz/ml/weka/arff.html`). Normally arff only stores on class attribute. We store multiple class attributes by placing all class attributes at the end of the attribute list. This means that for a d output dataset, the last d attributes are classes. To tell SMODeLKit the number of class attributes, modify the relation name with a "-c" parameter. Here is an example of how to do this for a 2-output dataset:

```
@RELATION 'myRelationName: -c -2 '
```

Notice the "-" sign before the 2. This is required to make our datasets compatible with Meka[1]. It means that the class attributes are at the end of the attribute list rather than the beginning. SMODeLKit does not support class attributes at the beginning of the attribute list, so the "-" sign is required.

If you prefer to not modify the relation name of your datasets, you can also specify the number of class attributes using the "-U" option when your run SMODeLKit.

SMODeLKit supports nominal and numeric attributes. It does not support string and date attributes. It does support instance weights.

4 Saving a Trained Model

When you train a model using the the "training" or "static" evaluation methods (see "-E"), then the learned model is automatically serialized and stored in the "models" folder. To use a serialized model later, use the "-D" option. Here is an example of training and then deserializing model:

[1] See `http://meka.sourceforge.net/`.

Learning Algorithm	Class	Settings File
Hierarchical Multi-Output Nearest Neighbor Model	HMONN	hmonn.json
Independent Classifiers	IndependentClassifiers	ic.json
Monolithic	MonolithicTransformation	monolithic.json
Multilayer Perceptron	NeuralNet	neuralnet.json
Ranked-CC	RankedCC	rankedcc.json
Wrapper for Weka	WekaWrapper	weka_SMO.json
Weighted Output Vector Ensemble	WOVEnsemble	wove.json
ZeroR	ZeroR	zeror.json

Table 1: A select of learning algorithms implemented in SMODeLKit.

```
java -cp 'bin:libraries/*' smodelkit.MLSystemsManager -L zeror model_settings/zeror.json \
 -A Datasets/mdc/synthetic/continuous_2out_4class.arff -E training

java -cp 'bin:libraries/*' smodelkit.MLSystemsManager -D ZeroR.ser \
 -A Datasets/mdc/synthetic/continuous_2out_4class.arff -E training
```

Notice that "-D" is relative to the "models" folder.

5 Implemented Learners

In this section we give a description of select learning algorithms of interest implemented in SMODeLKit. Table 1 gives a list of learning algorithms along with their class names and an example settings file for each. Documentation for each of these learning algorithms in in the corresponding Java files. Note that settings files are in the folder "model_settings". Documentation for the settings needed for each learner is also in the Java files, on one of the "configure" methods.

6 Weka Integration

SMODeLKit can use learners from Weka[2] for predicting single outputs, or within problem transformation techniques for predicting multiple outputs, such as with Ranked-CC.

By default Weka integration is disabled because enabling it would requre SMODeLKit to use Weka's GNU GPL license. If you choose to enable it, you must also abide by the terms of its license. To enable Weka integration, move WekaWrapper.java from the package "excludedFromBuild" to "smodelkit.learner". You will then need to link your build to weka in whatever way is convenient for the build tools you are using. For an example of the settings needed by WekaWrapper, see "model_settings/weka_SMO.json".

Weka integration can go both ways. If you wish to use SMODeLKit learners in Weka, use SMODeLKitWrapper.java, which is also in the package "excludedFromBuild". SMODeLKitWrapper is a Weka classifier, so to use it you will need to download the Weka source code and rebuild it with SMODeLKitWrapper.java. Weka only allows predicting a single output, so SMODeLKit learners in Weka will only be able to predict one output.

[2]See `http://www.cs.waikato.ac.nz/ml/weka/`.

SMODeLKit stores instances in memory in a way compatible with Weka instances, so there is very little memory overhead from using Weka learners in SMODeLKit. This is not true when using SMODeLKit learners in Weka.

7 Extending SMODeLKit

Learners, filters, and evaluators, are all plugins, so you can create new ones without recompiling SMODeLKit.

7.1 Creating a New Learner

To create a new learner, create a class which extends smodelkit.learner.SupervisedLearner. In order to load your learner as a plugin, it must have a no-arg constructor. Several of the abstract methods you will need to implement start with "canImplicitlyHandle...". These methods tell SMODeLKit what capabilities your learner has. If you give it a dataset which does not match these capabilities after all filters have been applied to it, an exception will be thrown.

There are multiple prediction methods your learner can override depending on its capabilities. Learners must override innerPredict, or innerPredictScoredList, or both. The method innerPredict is for predicting a single output vector, and innerPredictScoredList is for predicting a ranked list of output vectors with scores. Optionally, you can also override predictOutputWeights to predict scores (weights) for every nominal value in a single output vector.

All learners must have a settings file, even if it is empty. The settings file is for storing user-specified hyper parameters for you learner. To extract settings from a settings file, implement SupervisedLearner.configure(JSONObject). For an example of how to do this, see smodelkit.learner.NeuralNet.

To use your new learner, use its canonical class name as the first argument to the -L option when running MLSystemsManager.

7.2 Creating a New Filter

To create a new filter, extend smodelkit.filter.Filter. Make sure to create a no-arg constructor. To use the filter with a learner, create a "filter" json array in the learner settings file. See model_settings/neuralnet.json for an example. Use the canonical name of you filter class in place of the short names in neuralnet.json. Remember that filters are applied in the order in which they appear in the "filters" json array.

7.3 Creating a New Evaluator

To create a new evaluator, extend smodelkit.evaluator.Evaluator. Make sure to create a no-arg constructor. To use your evaluator, use the canonical class name with the -M option when running MLSystemsManager.