LearnPsdd Manual

Yitao Liang, Jessa Bekker

January 7, 2019

1 Using the PSDD library

1.1 Setting up dependencies

Add `lib` to `$LD_LIBRARY_PATH` and `$PATH`:

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:lib/
# export PATH=$PATH:lib/
```

Add this lines to ∼`/.bashrc` (or similar) if you want to permanently add it.

Check if metis and blossom are working:

```
# gpmetis
  Missing parameters.
   Usage:  gpmetis [options] <filename> <nparts>
           use 'gpmetis -help' for a summary of the options.
# blossom5
  Usage:  see USAGE.TXT
```

If a different text (*"No such file or directory"*) is shown, you need to download and compile it yourself, as explained in section 2.2.

1.2 Basic commands

Print possible commands:

```
# java -jar psdd.jar
```

- `learnPsdd BU`: bottom-up PSDD structure learning.

- `learnPsdd TD:` top-down PSDD structure learning.
- `learnPsdd search:` PSDD structure search.
- `learnEnsemblePsdd softEM:` structure learning of an ensemble of PSDDs by softEM.
- `sdd2psdd:` PSDD learning by doing parameter learning on an SDD.
- `learnParams:` Learn the parameters of a PSDD.
- `learnVtree:` Learn a vtree from data.
- `paramSearch:` Search for the best parameter calculator for a PSDD.
- `check:` Check if a PSDD is valid and calculate its likelihoods in two different ways.

By running any of these commands followed by `--help`, you get an explanation of the command and all its possible options. Try for example:

```
# java -jar psdd.jar learnPsdd search --help
```

Most of the options have default values that are good for most settings. In the following we introduce what need to be manually set for the the four most used functions: (i) learnVtree; (ii) sdd2psdd; (iii) learnPsdd; (iv)learnEnsemblePsdd.

learnVtree

Name	Abbr.	Value
`-d`	`--trainData`	*.train.wdata
`-v`	`--vtreeMethod`	the method to learn the vtree. The default is miBlossom.
`-o`	`--out`	Output vtree path. For example, if you want your output vtree to be in the folder "/exampleFolder/" and has the name "exampleVtree", then here you should put "/exampleFolder/exampleVtree". By doing so, if the method is chosen to be miBlossom, your vtree would be "/exampleFolder/exampleVtree_miBlossom.vtree"

sdd2psdd

Name	Abbr.	Value
`-v`	`--vtree`	Vtree file path, there is a folder with vtrees in the repo. They can be learned with `learnVtree`.
`-d`	`--trainData`	*.train.wdata
`-b`	`--validData`	*.valid.wdata
`-t`	`--testData`	*.test.wdata
`-o`	`--out`	Output PSDD path. For example, if you want your output PSDD to be "/exampleFolder/example.psdd", then here you should put "/exampleFolder/example.psdd".
`-m`	`--smooth`	Smoothing type, l-1 (Laplace smoothing) is usually a good one.
`-s`	`--sdd`	The sdd file where parameter leaning is performed upon.

learnPsdd

Name	Abbr.	Value
`-v`	`--vtree`	Vtree file path, there is a folder with vtrees in the repo. They can be learned with `learnVtree`.
`-d`	`--trainData`	*.train.wdata
`-b`	`--validData`	*.valid.wdata
`-t`	`--testData`	*.test.wdata
`-o`	`--out`	Output folder path, this should be an empty folder because multiple output files are created.
`-m`	`--smooth`	Smoothing type, l-1 (Laplace smoothing) is usually a good one.
`-p`	`--psdd`	Optional: Psdd file path. If a psdd is provided, then this is a base for learning, otherwise the learning is started from a PSDD with independent variables. To learn on top of constraints, the constraints in SDD format can be converted to a PSDD with `sdd2psdd`.

learnEnsemblePsdd softEM

Name	Abbr.	Value
`-v`	`--vtree`	Vtree file path, there is a folder with vtrees in the repo. They can be learned with `learnVtree`.
`-d`	`--trainData`	*.train.wdata
`-b`	`--validData`	*.valid.wdata
`-t`	`--testData`	*.test.wdata
`-o`	`--out`	Output folder path, this should be an empty folder because multiple output files are created.
`-c`	`--numComponentLearners`	The number of component learners used to construct the ensemble.
`-m`	`--smooth`	Smoothing type, l-1 (Laplace smoothing) is usually a good one.
`-p`	`--psdd`	Optional: Psdd file path. If a psdd is provided, then this is a base for learning, otherwise the learning is started from a PSDD with independent variables. To learn on top of constraints, the constraints in SDD format can be converted to a PSDD with `sdd2psdd`.

1.2.1 Hidden commands

There are two hidden commands that are useful during development: `scratch` and `assertTest`. The former executes the code in the scratch space in the Main.scala file. The latter checks if assertions are on or off.

```
# java -jar psdd.jar scratch
     Scratch output
# java -jar psdd.jar assertTest
 assertions are on
     or
 assertions are off
```

1.3 SBT

The scala build tool (SBT) is used for compiling, dependency management and jar creation. It can also be used to run and test the code. But it is recommended only to use it like this during development but to use the jar for actual usage. Any sbt command compiles the code first, so there is no need to do this explicitely.

To run the code, use the command `sbt ‘‘run parameters’’`. For example:

```
# sbt ‘‘run scratch’’

     Scratch output

# sbt ‘‘run learnPsdd search -d pathToDatasets/nltcs.train.wdata
-b pathToDatasets/nltcs.valid.wdata -t
pathToDatasets/nltcs.test.wdata -v vtrees/nltcs_miBlossom.vtree
-m l-1 -o nltcsOut’’

     Psdd structure learning
```

To test the code, run:

```
# sbt test
```

To build a jar, run:

```
# sbt assembly
```

This wil create the psdd.jar file in the folder `target/scala-2.11/`.

All the build settings are defined in `build.sbt`. In this file assertions can be turned on or off by commenting one of the “scalaOptions” lines.

1.4 Running experiments

When you use the code to run experiments, remember to:

- Turn off assertions: this is a lot of unnecessary overhead.
- Use the jar, running through sbt also creates overhead.

1.5 Output

Structure learning generates multiple output files:

- `out/progress.csv` keeps the learning progress. For each iteration, it saves the size, log likelihoods, and timings.
- `out/cmd` saves the command that was used to start this learning.
- `out/out` is unused for now
- `out/models` contains the models learned. It saves the psdd, vtree and dot file.
- `out/debug` contains debug information.

1.6 File formats

We use the same file formats as the SDD library.

Data files have no header and one line per (unique) example. An example is a comma seperated list of zeros and ones. If it only contains unique examples, then the line is preceded by a weight and a bar. For example, the following snippet contains 3 unique examples, 14 in total and has 8 variables:

```
8|1,1,0,1,0,1,0,0
4|0,1,1,1,0,1,0,1
2|0,0,0,0,1,0,1,0
```

PSDD files are similar to SDD files. They start with the number of nodes and all the following lines are nodes that appear bottom-up. There are three types of nodes:

- **Literals:** L id-of-literal-sdd-node literal
- **True nodes:** T id-of-true-sdd-node id-of-vtree trueNode variable log(litProb)
- **Decomposition nodes:** D id-of-decomposition-sdd-node id-of-vtree number-of-elements id-of-prime id-of-sub log(elementProb)*

We do not require the id's to start at 0 nor to be in order. This is for debugging purpose, so that we can map prints to the saved PSDDs.

2 Dependencies

Most of the dependencies are automatically added by `sbt` by either downloading the library or getting it from `lib/`. However, the `$LD_LIBRARY_PATH` and `$PATH` need to set manually by adding `lib/` to it. You may need to compile `gpmetis` and `blossom5` yourself.

2.1 The SDD library

The SDD library is used by the PSDDs to represent their internal formulas. For this purpose we use the `JSDD` Java wrapper of the original SDD C library. The required files are:

- `JSDD.jar`: The Java library, in `java.library.path`. This is set automatically by `sbt`.
- `libsdd.so`: The original SDD C library, in `$LD_LIBRARY_PATH`. This is *not* done automatically.
- `libsdd_wrap.so`: The native bridge between Java and C, in `$LD_LIBRARY_PATH`. This is *not* done automatically.

2.2 Libraries for learning vtrees

To learn vtrees, graph abstractions are used. For top-down learning, we use graph partitioning which is implemented by the `metis` library. For bottom-up learning, we use `blossom V`. Both libraries are called through system commands, therefore we need them in the `$PATH`. This is *not* done automatically.

Compiled versions of the libraries can be found in the `lib/` folder. However, they might not be right for your machine. If this is the case, you need to download them and follow their compilation instructions to compile them. The websites are:

- `http://glaros.dtc.umn.edu/gkhome/views/metis`
- `http://pub.ist.ac.at/~vnk/software.html`

Once they are compiled, the binaries in the `lib` folder are to replaced with the newly compiled ones:

- *`metisFolder`*`/build/Linux-x86_64/programs/gpmetis`
- *`blossomFolder`*`/blossom5`

2.3 Other libraries

Three external Java libraries are used:

- `Guava:` to implement the PSDD node cache.
- `Scopt:` to parse the command line options
- `Junit:` for unit testing

These libraries are automatically downloaded from their Maven repositories by `sbt`.

3 Understanding the code

To help understanding the code, we briefly explain all classes below and then explain the code flow of PSDD structure learning.

3.1 Classes

3.1.1 PSDD structure and operations

PsddNode is the core class for the PSDD structure. It is the super class for any type of PSDD node.

PsddElement is the element of a PSDD. Originally this was a triple: (prime, sub, θ). Here it is extended with the data of that element and it internal formula.

VtreeNode is the core class for the Vtree structure. It is the super class for any type of Vtree node.

PsddManager manages PSDD operations that affect their structure while keeping them valid and avoiding redundancy by keeping a node cache. Its public methods are:

- `newPsdd` constructs a new PSDD that represents a distribution over independent variables
- `readPsdd` reads a PSDD
- `readPsddFromSdd` reads an SDD to a PSDD
- `executeSplit` splits a PSDD element
- `simulateSplit` simulates a split and calculates the potential log likelihood gain and number of added edges
- `executeClone` clones a PSDD node
- `simulateClone` simulates a clone and calculates the potential log likelihood gain and number of added edges
- `calculateParameters` calculates the parameters of a PSDD given a parameter calculator

PsddQueries executes operations on a PSDD that do not affect their structure, such as:

- log likelihood
- size
- entropy
- probability of a partial variable assignment
- checking if a node is valid
- saving as PSDD, SDD or DOT file.
- getting all the nodes or elements of a PSDD in a certain order

Data represents the data efficiently as a bitset. It allows multiple operations.

3.1.2 Algorithms

These are the core classes for the PSDD structure learning and vtree learning:

Learner is the main class for learning the structure of PSDDs

OperationQueue keeps the best operation for each PSDD node. The operations are ordered on their quality. It also updates the queue when the structure of the PSDD is changed.

OperationFinder finds the best operation for a certain PSDD node by simulating possible operations on it.

VtreeLearner learns the vtrees.

3.1.3 Small calculations and keeping information

ParameterCalculator calculates parameters. There are multiple implementations.

OperationScorer scores an operation based on the resulting log likelihood gain and added edges.

Constraint represents a formula that can be used to split on. It allows several calculations such as the model count of a node restricted to this formula.

SaveFrequency specifies when a model should be saved (each k iterations and always or only keeping the best models)

OperationCompletionType specifies how a split or clone operation should be completed (minimal, complete, maximum k edges, maximum k depth)

CloneSpecification specifies how a node should be cloned. The manager puts this in PSDD nodes as bookkeeping during a clone operation (that involves multiple nodes).

PsddOperation represents a split or clone operation. It calls the methods of the manager.

SingleNodeOperation keeps an operation and some stats (score, delta size, delta log likelihood). It is used to keep operations in the operation queue.

3.1.4 Administration

Main parses the input and calls the disired classes.

Output manages all the output. E.g.: saving a psdd (as vtree, psdd and dot) or writing something to a debug or output file.

3.2 Code flow

The code is entered through `Main` that parses the input and then calls the `Learner`.

The `Learner` repeatedly tells the `OperationQueue` to update itself and return the next best `PsddOperation` to then execute the operation.

To update itself, the `OperationQueue` checks which nodes were affected by the last operation makes a new `SingleNodeOperation` for it. It sorts the operations based on their score. A `SingleNodeOperation` gets it's concrete operation by asking the `OperationFinder` to find the best `PsddOperation` for that node.

The `OperationFinder` finds the best `PsddOperation` for a node by simulating multiple operations and selecting the best one using the `OperationScorer`. To find splits, for every element of a node, it recursively updates a lists of

`Constraint`s on that element, by splitting the constraint into two mutually exclusive that together are equal to the original constraint. It starts from `NoConstraint` and updates them by conditioning on a variable. This algorithm is similar to decision tree learning. To find clones, it tries different subsets of the parents to make the clone for.

The `PsddOperation` executes and simulates operations by calling the `PsddManager`. It passes a `ParameterCalculator` along so that the manager knows how to calculate the parameters.