Manual for Auto-WEKA 0.2

Chris Thornton & Frank Hutter
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4, Canada
{cwthornt,hutter}@cs.ubc.ca

June 5, 2013

Contents

1 Introduction

This document is the manual for Auto-WEKA, a tool that performs compined algorithm selection and hyperparmeter optimization over the classifiers implements in WEKA. More specifically, given a specific dataset, Auto-WEKA explores hyperparameter settings for a number of different classification methods with feature selection and recommends to a user which method will likely have good performance on a new set of never before seen data, using Sequential Model Based Optimization (SMBO) techniques.

This version of Auto-WEKA is research quality code, there are a few assumptions on the typical use case of Auto-WEKA (such as the location of certain configuration files) that will be removed in future versions. Additionaly, Auto-WEKA has been designed in mind for doing large numbers of experiments to see how effective the entire process can be, so it may seem like there is lots of overhead for someone who just has a simple problem the want to optimize. We are working at making this process much smoother, and any help in fixing these problems is greatly appreciated!

1.1 License

Auto-WEKA is open source software issued under the GNU General Public License

1.2 System Requirements

Auto-WEKA itself requires only Java 6 or newer to run, while the underlying optimizers that Auto-WEKA uses may have other requirements. Auto-WEKA has only been tested on Unix-compatible operating systems. Auto-WEKA can in theory use any version of WEKA, but it has been targeted against 3.7.9. There are a few minor changes that should be made to the WEKA classifiers, detailed inside the `autoweka.patch` provided. You can apply the patch to your WEKA distribution by running `patch -Np1 < autoweka.patch` from the weka source directory. The changes in this patch just add a support for classifiers to detect if the thread that they've been running in has been interrupted, and then break out of their training phase at their earliest convenience. We've provided a pre-compiled version of WEKA with the patches applied for you to use if you do not wish to compile your own version.

Note that you can still use an unmodified version of WEKA, just that you will likely get inferior performance, since if you ever encounter a classifier that takes more than your allotted time budget, it will be equivalent to a classifier that wasn't able to get a single correct result.

1.3 Version

This version of the manual is for Auto-WEKA 0.2.

2 Auto-WEKA Overview

Using Auto-WEKA can be broken down into three main steps. First, you have to build your experiment definition, which tells Auto-WEKA what dataset(s) to run on, as well as what kind of hyperparameter search will be done (either through an SMBO method, or through something like grid search). Once the definition has been written, the experiment needs to be fully instantiated by having Auto-WEKA detect what kind of classifiers can be used given the definition you wrote. Details about how to run experiments are covered in Section 3.

Once an experiment has been produced, it actually has to be executed. Auto-WEKA takes advantage of multiple cores by running the same experiment with different random seeds, the only requirement is that all the experiments have a similar file system (since Auto-WEKA relies on some absolute path names). Details about how to run experiments are covered in Section 4.

After the experiment has been executed, the analysis phase occurs. When Auto-WEKA uses an SMBO method, it produces a 'trajectory' of hyperparameters that were identified by the SMBO method as being the best at a particular point in time. Additional experiments can be performed on these trajectory points, for example to see if all the trajectory points have a similar performance on a new set of data that the SMBO method did not have access to. Details about how to run analysis experiments are covered in Section 6.

When running any of the classes for Auto-WEKA, you must have both `weka.jar` and `autoweka.jar` on the classpath. To simplify this process, there is a script named `autoweka` that will set up java for you, then pass any of the remaining arguments through to java.

---

3 Defining Experiments

Auto-WEKA can define experiments either on the command line through a rather tedious process (only documented in the code), or 'experiment batches' can be made. An experiment batch is an XML file that contains a list of datasets to perform experiments on, and one or more experiment prototypes that contain information such as the type of SMBO method to use or the way to partition the provided training data. These files were designed for performing large comparisons of different settings on multiple datasets, but they can easily be applied to situations where you have a single dataset. Section 3.1 provides a description of how to write these files, but you can also look at the `autoweka.ExperimentBatch` class and its JavaDocs to see how to use them. These experiment definitions also select the choice of SMBO method (see section 3.3), and the way to partition the training data using an 'InstanceGenerator' (see section 3.2).

Once the experiment definition has been created, Auto-WEKA needs to know what type of classifiers and feature/attribute selection methods to used. These are all specified in `.param` files, detailed in section 3.4.

Once the experiment definition is fully written, the main method of `autoweka.ExperimentConstructor` can be invoked on the experiment definition file to determine which classifiers can be used on your datasets, and will generate the files needed under the `experiments` folder from the current working directory.

Using the provided script, this command can be executed as follows:
`./autoweka autoweka.ExperimentConstructor path/to/experimentbatch.xml`

3.1 Experiment Definition Files

Experiment definitions are done in XML files, with a root node of an `experimentBatch`. Inside the experiment batch, there can be any number of `datasetComponent` and `experimentComponent` elements - this allows for quickly making experiments with multiple different settings on a variety of datasets. You can produce a template file by running the main method `autoweka.ExperimentBatch template.xml`.

3.1.1 datasetComponent

The `datasetComponent` defines a dataset that you want to perform experiments on. Auto-WEKA can take two formats for input. The first requires that datasets are a compressed zip containing exactly two files, `train.arff` and `test.arff`. Auto-WEKA will perform all of its experiments on the training data in `train.arff`, and will use the testing data in `test.arff` for analysis after the experiment has completed. (If you don't have any testing data, you can just create a dummy dataset file). A datasetComponent has the following form in the XML file with self explanatory names

```
<datasetComponent>
  <zipFile>path/to/zip/file.zip</zipFile>
  <name>DatasetName</name>
</datasetComponent>
```

Additionally, Auto-WEKA can be pointed to the `train.arff` and `test.arff` individually. These datasetCompoments have the following form

```
<datasetComponent>
  <trainArff>path/to/train.arff</trainArff>
  <testArff>path/to/train.arff</testArff>
  <name>DatasetName</name>
</datasetComponent>
```

3.1.2 experimentComponent

The `experimentComponent` element contains the parameters that you want to use in an experiment on all of the `datasetComponent`s defined in the same file.

```
<experimentComponent>
  <name>SMAC-CV10</name>
  <resultMetric>errorRate</resultMetric>
  <experimentConstructor>autoweka.smac.SMACExperimentConstructor</experimentConstructor>
  <instanceGenerator>autoweka.instancegenerators.CrossValidation</instanceGenerator>
  <instanceGeneratorArgs>numFolds=10:seed=0</instanceGeneratorArgs>
  <tunerTimeout>108000</tunerTimeout>
  <trainTimeout>9000.0</trainTimeout>
  <attributeSelection>true</attributeSelection>
  <attributeSelectionTimeout>900</attributeSelectionTimeout>
  <memory>3072m</memory>
  <extraProps></extraProps>
</experimentComponent>
```

name The name of the experiment component, this will be combined with the data set when the full experiment is created.

resultMetric The resultMetric to use. Most likely `errorRate` for classification. `rmse`, `rrse`, `meanAbsoluteErrorMetric` and `relativeAbsoluteErrorMetric` are also supported. See the source for `autoweka.ClassifierResult` for more.

experimentConstructor The name of the class to be used to build the experiment - see 3.3 for a list of what values are supported.

instanceGenerator The name of the class to be used for partitioning the training data, see 3.2 for what classes are implemented.

instanceGeneratorArgs A property string (`var1=val1:var2=val2...`) with arguments to the instance generator.

tunerTimeout The number of seconds to run the SMBO method.

trainTimeout The number of seconds to spend training a classifier with a set of hyperparameters on a given partition of the training set.

attributeSelection `true` if you want to consider using different feature/attribute selectors, `false` otherwise (or don't include it).

attributeSelectionTimeout Number of seconds to spend doing feature/attribute selection.

memory The memory limit that is passed to the Java instance that is training the classifier

experimentProps Optional extra arguments that can be passed to an experiment

allowedClassifier Optional argument that restricts what classifiers can be considered. Specify it multiple times to include a subset of classifiers, or don't specify it at all to make Auto-WEKA chose all possible classifiers.

It is also possible to indicate in an experiment if some extra computations should be done on all points in the trajectory. In your `experimentComponent`, you can create any number of `trajectoryPointExtras`:

```
<experimentComponent>
  <name>humanReadableName</name>
  <instance>instanceString</instance>
</experimentComponent>
```

Here, the `name` is a human readable name the corresponds to the `instanceString` that is being executed. The instance string is what is passed to the instance generator, covered in section 3.2.

3.2 Instance Generators

Auto-WEKA uses an instance generator to partition the training and testing data provided into new sets of training and test data for the SMBO method to use, e.g. the training data is broken up into 10 folds in cross validation. Each partition of the data is called an 'instance', and can be specified through a string, often in the form of a property string (`var1=val1:var2=val2:...`). Additionally, each Instance Generator also has a string of `instanceGeneratorArgs` that determine what instances will be created in your experiment. For the most common instance generators Auto-WEKA provides, these strings are detailed below.

The instance string `default` however has a the special meaning that the training and testing data are unmodified.

3.2.1 Default

The most boring of instance generators - it does nothing to the input training and test data, it ignores any arguments and returns the unmodified partition of instances.

3.2.2 Cross Validation

Performs k-fold cross validation on the training set. Implemented in `autoweka.instancegenerators.CrossValidation`.

Generator Args:

A property string containing the following two elements:

seed The seed to use for randomizing the dataset

numFolds The number of folds to generate

Instance String

A property string containing the following three elements:

seed The seed to use for randomizing the dataset

numFolds The number of folds total

fold The instance's fold number

3.2.3 Random Sub-Sampling

Performs generates an arbitrary number of folds by randomly making a partition of the training data of a fixed percentage. Implemented in `autoweka.instancegenerators.RandomSubSampling`.

Generator Args:

A property string containing the following the following elements:

startingseed The seed to use for randomizing the dataset

numsamples The number of subsamples to generate

percent The percent of the training data to use as 'new training data'

bias Optional: The bias towards a uniform class distribution

Instance String

A property string containing the following three elements:

seed The seed to use for randomizing the dataset

percent The percent of the training data to use as 'new training data'

bias Optional: The bias towards a uniform class distribution

3.2.4 Termination Holdout

A meta instance generator that removes a random percentage of the training data before passing it on to the target instance generator. Implemented in `autoweka.instancegenerators.TerminationHoldout`.

Generator Args:

A string with three components, separated by `[$]`. The first component contains the parameters for the holdout method (described below), while the second component contains the name of the target instance generator that will receive the subsampled dataset. The final compoment contains all the arguments that will be passed on to the generator of the target A property string containing the following the following elements:

terminationSeed The seed to use for randomizing the dataset

terminationPercent The percent of the training data to use hold back

terminationBias Optional: The bias towards a uniform class distribution

Instance String

A string with three components, separated by `[$]`. The first component contains the parameters for the holdout method (described below), while the second component contains the name of the target instance generator that will receive the subsampled dataset. The final compoment contains all the arguments that will be passed on to the generator of the target

terminationSeed The seed to use for randomizing the dataset

terminationPercent The percent of the training data to use as 'new training data'

terminationBias Optional: The bias towards a uniform class distribution

3.3 SMBO Method

Currently Auto-WEKA supports two different SMBO methods, the Tree based Parzen Estimator (TPE) and Sequential Model-based Algorithm Configuration (SMAC). Each method requires some initial set up with Auto-WEKA so that it can be used smoothly, namely by creating `.properties` files that tell Auto-WEKA where to find each method (These files must be in the current working directory when you invoke any of the experiment constructor commands. The syntax of a properties file is of the form `var=value`, with one variable per line). Each of the following sections mentions how to tell Auto-WEKA to use SMBO method, as well as the name and contents of the `.properties` file that must be created.

3.3.1 SMAC

SMAC was designed for algorithm configuration, but can easily be used in other cases of black box optimization. Auto-WEKA requires a version 2.04.01 of SMAC or greater. To build an experiment with SMAC, you use the `autoweka.smac.SMACExperimentConstructor`.

autoweka.smac.SMACExperimentConstructor.properties

`smacexecutable` The path to the `smac` script inside the SMAC distribution.

A few of SMAC's options are also supported by Auto-WEKA, look inside the `autoweka.smac.SMACExperimentConstructor` class to see what variables are supported in the `extraProps` of an experiment definition.

3.3.2 TPE

The TPE is provided by the Hyperopt project, written for Python 2.7. To build an experiment with TPE, you use the `autoweka.tpe.TPEExperimentConstructor`.

autoweka.smac.SMACExperimentConstructor.properties

`python` Optional: The Python you want to use - defaults to trying to find a `python` on the system path.

`pythonpath` Optional: Sets the PYTHONPATH environment variable before invoking python (useful if you don't have hyperopt inside your site-packages).

`tperunner` Path to the `tperunner.py` file in the Auto-WEKA source directory

3.4 Parameter Files

Auto-WEKA groups classifiers into 3 categories: base, meta and ensemble. Meta classifiers are methods that take a single base classifier and use it to perform classification (like AdaBoost), while ensemble methods use a number of base classifiers to perform classification. Additionally, Auto-WEKA supports feature/attribute selection, through the use of search and evaluator methods. Many of these methods have parameters that influence their behaviour, and these parameters are exposed through the use of `.param` files. The name of the file contains the full class name of the method that we are exposing to Auto-WEKA (so for WEKA's SVM implementation, the file would be called `weka.classifiers.functions.SMO.params`), and it is placed inside the subfolder of the `params` directory corresponding to the method (so the SVM implementation goes inside the `base` subfolder, while the param file for AdaBoost would go inside the `meta` subfolder).

The contents of these files can be broken down into two parts: parameter definitions and conditional statements. For examples for a number of different classifiers and feature selection methods, see the the provided `.param` files that come with Auto-WEKA.

3.4.1 Parameter Definitions

For each parameter exposed to Auto-WEKA is written on its own line, and has the following form.

`<FLAGS>_<NAME> <DOMAIN> <DEFAULT><TYPEFLAGS>`

`FLAGS` These will all be stripped by Auto-WEKA before they are passed to WEKA. The flags are defined up until the last underscore before the `NAME`. Summary of valid flags:

`HIDDEN` The parameter will be never be seen by the WEKA method.

`INT` The parameter will be forced to an integer when it is passed on to WEKA.

`QUOTE_START` A quote character will be inserted after this parameter, up until the next `QUOTE_END`

`QUOTE_END` Inserts a quotation character and removes the parameter

`DASHDASH` Inserts a double dash (`--`) into the call string

`NAME` The name of the argument that WEKA expects on the command line. Note that in most cases, these are single capital letters (and never contain an underscore)

`DOMAIN` The domain of this parameter, which is either numeric or categorical.

Categorical The domain is specified as a comma separated list of strings in between two curly braces, eg. {`v1, v2, ... vk`}

Numeric The domain is specified as two comma separated numbers for the lower and upper range of the domain between two square brackets, eg. `[0.1, 10]`

`DEFAULT` The default value of the parameter (which must be inside the domain) is specified between two square brackets eg. `[1.0]`

`TYPEFLAGS` Optional type flags for numeric domains. If you want to recommend to the SMBO method that this parameter should be treated as an integer, add an `i`. If the parameter should be sampled on a logarithmic scale, add a `l`

Additionally, Auto-WEKA defines special treatment to the categorical domain of {`REMOVED, REMOVE_PREV`}, which can be best demonstarted through an example. Suppose we have a parameter `M` which is a flag of a classifier that enables aggressive memory caching. If `M` is set to `REMOVED`, by Auto-WEKA, then WEKA will receive the argument `-M`. In the case that `M` is set to `REMOVE_PREV`, then Auto-WEKA will completely hide the `-M` flag from the WEKA classifier.

Note: Arguments are sorted alphabetically by Auto-WEKA before they are passed on to the WEKA method.

3.4.2 Conditionals

For some classifiers, only some parameters make sense once another parameter takes on a certain value. If this is the case, after all the parameters have been defined in the `.param` file, you need to inform Auto-WEKA of these conditionals. All conditionals must appear after a `Conditionals:` line. The format of a conditional line is as follows

```
<PARAMETER> | <PARENT> in {<VALUE1>, <VALUE2>, ...}
```

`PARAMETER` The name of the child parameter that is active based on the value of the parent

`PARENT` The name of the parent parameter that the conditional depends on

`VALUE*` If the parent parameter takes on one of the values in this list, then the child parameter will be enabled, Otherwise, the child parameter is disabled.

4 Running Experiments

After generating your experiment using the main method of the `ExperimentConstructor`, execute the main method of `autoweka.Experiment` with two parameters - the path to the experiment folder and an initial seed for the random number generator of the SMBO method. Auto-WEKA has been designed to execute many optimization runs in parallel, simply change the seed that you pass to each invocation of `autoweka.Experiment`. Auto-WEKA will now grind away for a while until the `tunerTimeout` has been hit as specified in the experiment definition.

Once your generate the experiment by running the `ExperimentConstructor`, Auto-WEKA tries to resolve path names fully, so it is unlikely that you can move these folders around and still run the experiment.

---

5 Analyzing Experiments

After an experiment finishes, Auto-WEKA produces a trajectory of good looking classifiers and hyper-parameters inside the experiment subdirectory, with the naming scheme of `<ExperimentName>.trajectories.<Seed>`. Once all of your experiments have completed, each of these files needs to be merged into a single trajectory group for analysis. Run the main method of `autoweka.TrajectoryMerger`, with a single argument of the experiment's directory. This produces a single file `<ExperimentName>.trajectories` inside the experiment's folder.

If you defined any `trajectoryPointExtras` in your experiment definition, you'll want to run the main method of `autoweka.TrajectoryPointExtraRunner` with the arguments `<ExperimentFolder>/<ExperimentName>.trajectories.<Seed>` before you run the `TrajectoryMerger` - this does runs the classifer/hyperparameters identified in the trajectory on whatever instance strings you've specified.

Finally, to get the best hyper-parameters and classifier that Auto-WEKA has found on the dataset, run the main method of `autoweka.GetBestFromTrajectoryGroup`, with the single command line argument pointing at the `.trajectories` file that was produced in the last step. This will print out information on how many trajectories were used to select the best, what Auto-WEKA thinks the performance of the selected method on the training set is, as well as which classifier was chosen and the command line arguments that should be given to the classifier.

6 Sample Experiment Walkthrough

Include in the distribution is a sample experiment definition that runs SMAC on the German Credit dataset from the UCI repository, this section will show you how to run a typical Auto-WEKA experiment.

First, navigate to the `autoweka` directory that contains the `autoweka.jar` file. First, we need to build the actual experiment by running the `ExperimentConstructor`

For all the following Java commands, we assume that you run them inside the `autoweka` directory, and add both `autoweka.jar` and `weka.jar` to the Java's class path.

```
java autoweka.ExperimentConstructor sampleexperiment.xml
```

This will load the dataset in `sampledata/creditg.zip`, determine what classifiers and feature selectors that are defined in the `params` directory can be used, and write out the experiment into the `experiments` directory.

Note that you will have to modify `autoweka.smac.SMACExperimentConstructor.properties` to point to your distribution of SMAC, specifically a script that loads all the needed jars and invokes the main method.

Next, we need to run our experiments by invoking the `Experiment` with different seeds

```
java autoweka.Experiment experiments/SMAC-CV10-GermanCredit 0
java autoweka.Experiment experiments/SMAC-CV10-GermanCredit 1
java autoweka.Experiment experiments/SMAC-CV10-GermanCredit 2
...
```

After watching some paint dry (and the experiments have completed), we now need to combine all the trajectories into a single file using the `TrajectoryMerger`

```
java autoweka.TrajectoryMerger experiments/SMAC-CV10-GermanCredit
```

If you've defined a number of `trajectoryPointExtras` in your experiment definition, you'll want to invoke

```
java autoweka.TrajectoryPointExtraRunner \
            experiments/SMAC-CV10-GermanCredit/SMAC-CV10-GermanCredit.trajectories.0
```

for each completed trajectory before you run the merger.

Now, you can perform any kind of analysis on this merged trajectory file (future versions of Auto-WEKA will come with a Python API for working with these trajectory groups), but the most common operation you'd want is to find the classifier/hyperparameters that have the best performance using `GetBestFromTrajectoryGroup`

```
java autoweka.GetBestFromTrajectoryGroup \
           experiments/SMAC-CV10-GermanCredit/SMAC-CV10-GermanCredit.trajectory
```

7 Developer Comments

Auto-WEKA has been designed to be relatively easy to extend with new SMBO methods/instance generators/classifiers. The core classes in Auto-WEKA are all have JavaDoc, and comments throughout the code that should help explain what each bit does.

To add a new SMBO method, you need to provide three classes, an `ExperimentConstructor`, a `TrajectoryParser`, and a `Wrapper`. The `ExperimentConstructor` converts an experiment definition to an actual experiment file (by generating any extra data that is needed by the SMBO method), while the `TrajectoryParser` extracts the results of the SMBO method into a format that can readily be used by the rest of the Auto-WEKA tools. The `Wrapper` class provides a way to convert parameters from the SMBO method into something that can be understood by Auto-WEKA. This class is also responsible for reporting the error rate back to the SMBO method, along with the time it took to train the classifier. Looking at the two provided implementations for SMAC and TPE should be sufficient in determining how to write your own methods.

New instance generators can be created by extending `autoweka.InstanceGenerator`, and just ensuring that they are on the classpath when you invoke the `ExperimentConstructor`. Looking at the provided generators should be sufficient for creating your own, (which would allow you to build generators that don't require the entire dataset loaded into RAM).

Adding a new classifier into Auto-WEKA is as simple as creating a new `.param` file in the appropriate subfolder under the `params` directory, and ensuring that your classifier is on Java's Classpath when you invoke the `ExperimentConstructor`.

More coming soon