

[Manual Home](#)

# Magpie Tutorial

This tutorial is designed to explain how to use Magpie to make a machine-learning-based model for the formation energy of crystalline compounds based on composition. It covers installation briefly, launching Magpie from the command-line, preparing data in a Magpie-friendly format, and the basics of creating and using models.

## Installing Magpie

Before installing Magpie, you need to make sure your system has the Java Runtime Environment Version 7 or greater. To do so, open up your computer's command-line prompt and call `java -version`. If the first line of the output doesn't look like `java version 1.7_071`, go to [Java.com](http://Java.com) and install the latest version.

Once your computer has the correct version of Java, download the latest version of Magpie from [OQMD.org](http://OQMD.org) and extract it. The ZIP file available from this link is updated nightly. This folder includes a compiled version of Magpie, this documentation, and a few example scripts.

Alternatively, you can install the latest copy of Magpie by cloning the git repository on [bitbucket.org](http://bitbucket.org) and building Magpie yourself. See [the installation guide](#) for more details.

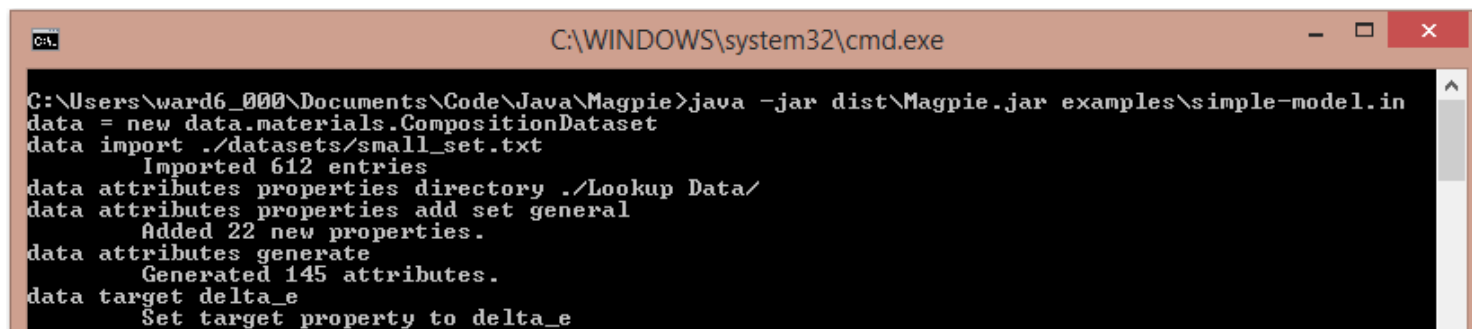
To verify that everything works, open a command prompt and navigate to your new Magpie folder, and then call

```
java -jar dist/Magpie.jar
```

(Note: If you are using the Windows Command Prompt, the command to launch Magpie is `java -jar dist\Magpie.jar`).

This should open an interactive prompt for Magpie. Press "Enter" or type "exit" to close this prompt.

Another route for using Magpie is to write input files that contain a script describing what Magpie should do. Launch Magpie with the example input script [examples/simple-model.in](#) by calling `java -jar dist/Magpie.jar examples/simple-model.in`. You should see the echos of the commands in the input file and output being printed to screen. If so, Magpie is ready to run on your system.



```
C:\WINDOWS\system32\cmd.exe

C:\Users\ward6_000\Documents\Code\Java\Magpie>java -jar dist\Magpie.jar examples\simple-model.in
data = new data.materials.CompositionDataset
data import ./datasets/small_set.txt
  Imported 612 entries
data attributes properties directory ./Lookup Data/
data attributes properties add set general
  Added 22 new properties.
data attributes generate
  Generated 145 attributes.
data target delta_e
  Set target property to delta_e
```

## Formatting and Importing Data

Magpie expects whitespace-delimited input files where the first line is a header describing the data (e.g., property names) and the first column is a string describing the material. For example, a dataset containing the composition and formation energy of materials could look like

```
composition delta_e stability{Yes,No}
NaCl -5 Yes
Fe2O3 -4.2 Yes
Ni3.00Al1 -0.4 Yes
Mg(NO3)2 None No
```

A few key things to note about this example data file are that the acceptable format for the composition is broad, "None" can

be listed if a measurement is not available, and it is possible to define categorical properties by listing the category names in {}'s after the property name. Further details of input file formats are described in the Javadoc for Magpie (ex: see [CompositionDataset](#)).

To start this tutorial, create a text file named "tutorial-part-1.in". Over the next sections, we will add to commands to this file that generate all of the required components for a machine learning model.

The first commands in the input file, will load data into Magpie by first creating a variable to store the data and then calling the "import" command for that variable. As in the [examples/simple-model.in](#) example (which is the basis of the tutorial), composition data is loaded in from a sample dataset using the commands:

```
data = new data.materials.CompositionDataset
data import ./datasets/small_set.txt
```

The first command creates a variable representing a CompositionDataset object and names it "data." All of the available commands of this variable are listed [here](#). In general, you can find the available commands for any variable type from [the Variables documentation page](#). As described in the referenced documentation pages, the "import" command of data is called with the path of the dataset file as an argument, as shown in the second command.

After running these commands, the composition and measured properties of each of the materials described in "small\_set.txt" are stored in data. The "small\_set.txt" dataset contains some entries with the same composition, which we should resolve before building a machine learning model. In this tutorial, our goal will be to predict the formation enthalpy of the ground state structure at a composition. To do so, we should process the dataset so that only the lowest energy entry at each composition will be used to train the model. This can be accomplished by calling the "duplicates" command of data:

```
data duplicates RankingDuplicateResolver minimize PropertyRanker energy_pa SimpleEntryRanker
```

This command finds all duplicate entries (duplicate == has same composition) in data and selects the entry with the lowest value of the "energy\_pa" property among each group of duplicate entries. Incidentally, this command demonstrates several important characteristics of commands in Magpie. As before, the first word in the command is the name of a variable and the second is a command word. As described in the documentation for CompositionDataset (see [here](#) and [here](#)), the options for the "duplicates" command are

**duplicates <resolver> [<resolver options>]** – Eliminate duplicates within a dataset  
*resolver*: Name of [BaseDuplicateResolver](#) used to handle duplicates  
*resolver options*: Any options for the resolver

The first option of the duplicates command asks for the name of a [BaseDuplicateResolver](#) - a type of variable in Magpie - to be used to handle groups of duplicate attributes. In our example, the [RankingDuplicateResolver](#) is used for this option. As described in the [Javadoc](#), this resolution strategy ranks entries based on their measured properties and selects the duplicate with the best value of that rank. Following the options described in "Usage" section of the Javadoc page for [RankingDuplicateResolver](#), we specify that we want to select the minimum formation enthalpy using the options: [minimize PropertyRanker](#) energy\_pa [SimpleEntryRanker](#). I would recommend reading through the documentation for [PropertyRanker](#) in order to understand the options to this command. Under the hood, Magpie will create a RankingDuplicateResolver, set its options according to what we specified here, and run it on our dataset.

The next step in our script is to specify that the formation energy (which is named "delta\_e" in the data file) is what we are looking to model. To set this property as the class variable, run the "target" command:

```
data target delta_e
```

At this point, we have specified the output for our machine learning model and now need to generate the input variables: attributes. By default, a CompositionDataset will compute attributes described in a recent paper by [Ward et al.](#), which include attributes based on the properties of the constituent elements. To compute attributes it is therefore necessary to define where the elemental property lookup tables are located and which elemental properties should be considered when computing attributes. That is accomplished by calling two "attributes" commands of the data variable:

```
data attributes properties directory ./lookup-data/
data attributes properties add set general
```

Once these settings are defined, attributes are computed by calling:

```
data attributes generate
```

The "data" variable now contains 145 attributes describing the composition and the measured formation energy as the class variable. This information can be saved to disk using the save command. As described in the documentation for the [text interface](#), this command takes the name of the variable as the first argument, the desired filename as the format as the second, and (optionally) the desired format as the third. To save in CSV format, the command is

```
save data delta_e csv
```

This save command will generate a file named "delta\_e.csv". The CSV format in Magpie contains the attributes for each entry as the first N - 1 columns, and the value of the property being predicted as the last. This format is useful for generating datasets for use in analysis packages such as Weka (see next section), but is not the only format available in Magpie. The "json" format (saved by running `save data delta_e json`) contains detailed information about the dataset that includes the names of each attribute, composition of each entry, and the measured and ML-predicted values for each property for each entry. Different types of datasets have different save formats, which are described in pages available in [the Variables documentation page](#).

At this point, call Magpie to run this script by calling `java -jar dist/Magpie.jar tutorial-part-1.in`. If this script completes successfully, type `exit` to exit out of Magpie.

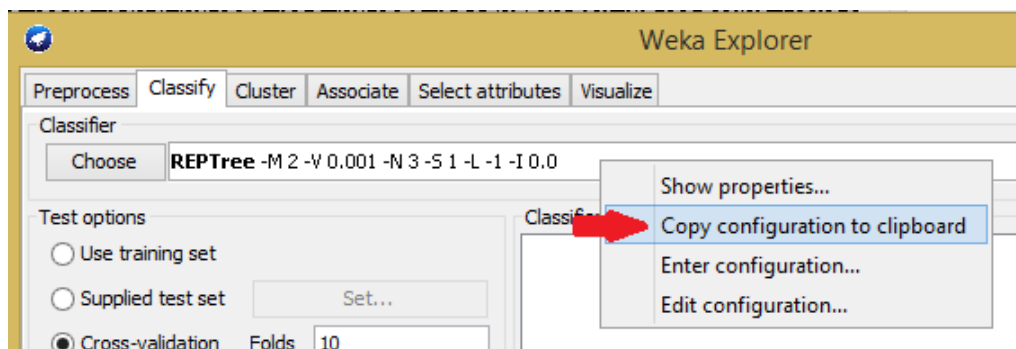
**Note** A complete script file for this section and the next is provided in [./doc/tutorial-files/tutorial-part-1.in](#)

## Building a Model

If you have done the previous parts of the tutorial, you now have a file named "delta\_e.csv" that contains the attributes and formation energy of a few hundred crystalline compounds. In this part of the tutorial, we will describe one method for finding a suitable machine learning algorithm for this data and creating a model in Magpie.

Most of the machine learning algorithms available through Magpie are provided by other software pages, such as [Weka](#) and [scikit-learn](#). For simplicity, this tutorial only describes how to use Weka and will only briefly skim over the features of Weka. If you want to learn more about these packages, it is strongly recommend to read the Weka documentation and, potentially, [the textbook](#).

Weka provides an excellent graphical interface for testing the performance of the wide variety of available algorithms. Again, for the purpose of brevity, this tutorial assumes that you have learned how to import data and run models in Weka. Once you have selected the algorithm that works best for your materials problem, the only information you need to save is the name of the algorithm and the desired settings. Luckily, Weka makes this easy. Simply opposite click on the name of the model and select "Copy configuration to clipboard".



The appropriate variable type for regression models using Weka is [WekaRegression](#). As discussed in the variable description page for WekaRegression (see [here](#)), the "Usage" for this command is the name of a Weka algorithm followed by the settings for the algorithm. To create a Weka model, add the following lines to the input file created in the previous part:

```
model = new models.regression.WekaRegression weka.classifiers.trees.REPTree &
-M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0
```

**Note** The command is split on to two lines using the "&" to mark that the lines should be combined together.

To train this model, add the training command

```
model train $data
```

The train command for variables that represent models will train the model using a dataset stored in another variable. When a variable is used in a Magpie command, it is accessed by putting a "\$" in front of the name of the variable (e.g., "\$data" to access the variable data).

Once the model is trained, you can print out the training statistics (which are automatically computed) using the print command. Like the save command, the first argument for print is a variable name, which is followed by the desired print command. To print training statistics of the variable "model", this is:

```
print model training stats
```

Similarly, one can perform 10-fold cross-validation and print the validation statistics by the two commands:

```
model crossvalidate $data 10  
print model validation stats
```

This model can be saved into a system-independent format by calling the save command without any format argument. It is also necessary to save data, which contains the settings necessary to generate the attributes, in order to use this model. As the training data is not needed to run the model, one can save a copy of data that does not contain the entries with the template format. The commands to do this are

```
save model delta_e-model  
save data delta_e-data template
```

After running this modified script, Magpie will save two files, delta\_e-model.obj and delta\_e-data.obj, that contain all of the information necessary to use your model and can be run on any system with Magpie installed.

## Searching for New Materials

If you have completed the other steps of this tutorial, you now have two Magpie object files (named delta\_e-model.obj and delta\_e-data.obj) that can be used to compute the formation energy of crystalline compounds. If you have not, run the [./doc/tutorial-files/tutorial-part-1.in](#) script.

The first step for using this model to find new materials is to create a dataset in which to store the search space. To do so, first launch Magpie and load in the object stored in delta\_e-data.obj using the load command:

```
search = load delta_e-data.obj
```

This command creates an empty dataset named "search." While this dataset does not contain any entries, it does contain all of the settings necessary to compute the same attributes used to train the model.

The next step is to generate a search space using the [IonicCompoundGenerator](#). As described in the "Usage" statement in the [Javadoc](#), this entry generator takes 4 arguments: the minimum and maximum number of constituents, the maximum number of atoms per formula unit, and a list of elements to use. The Magpie command to generate all ternary ionic compounds with less than 10 atoms per unit cell composed of Li, Fe, Ni, Zr, O, or S is:

```
search generate IonicCompoundGenerator 3 3 10 Li Fe Ni Zr O S
```

This should have generated a search space of 154 compounds. Next, you will need to compute attributes for these new entries. Since the search variable contains all of the settings for computing the attributes used when creating the formation energy model, you can call the "attributes generate" command without first specifying those options.

```
search attributes generate
```

Now, load and run the model by calling:

```
model = load delta_e-model.obj  
model run $search
```

After running the model, the predicted values for each entry are stored in search. Saving the data to disk in the "stats" format,

will produce a file with the compositions and predicted formation enthalpy for each compound

```
save search predicted-compounds stats
```

It is also possible to find entries with the greatest to least formation enthalpy

```
search rank 10 minimum predicted SimpleEntryRanker
```

If you used the REPTree algorithm shown earlier in the tutorial, Magpie should identify ZrSO and several other compounds as having the formation enthalpy at -2.978 eV/atom. Many entries having the same predicted value is a result of selecting a decision tree.

**Note** A complete script file for this section is provided in [./doc/tutorial-files/tutorial-part-2.in](/doc/tutorial-files/tutorial-part-2.in)

Many more examples for how to employ Magpie are available [in this documentation](#), which include scripts to determine which attributes correlate best with a certain material property and building machine learning models using Scikit-learn.