Daimensions™ How-to Guides

Introduction

Brainome's Daimensions™ is a tool that will help you:

- Understand your data
- Evaluate the likelihood of success in building a machine learning model from your data
- Guide your investments so that you spend the right amount of time and money in building your machine learning model
- Provide transparency during the process of building a machine learning model so that you can have a clear understanding of model performance
- Deploy easily, without the need for complicated "MLOps" frameworks and experts
- Keep your deployed model on track and functioning properly as operating conditions evolve

To get the most value from Daimensions™, follow the steps laid out in our How-to Guides:

Note: **terms in bold** are described in further detail in the **Daimensions™ Glossary**.

Prepare

The first step is to think about the big picture: What do you want to predict? Why is this important? Into what kind of workflow are you going to deploy this predictor? What are the consequences of correct or incorrect predictions in your target context? Is it more important that predictions be accurate or is it more important that they adapt properly to changing operating conditions?

Having thought about these things in advance will help you a great deal as you progress through the steps we lay out below.

Output classes

When working with Daimensions™, you'll always be doing **classification**. This is, your predictor will always be placing its output into one of many categories. It's up to you to decide what categories you need. For example, in binary classification you are asking a predictor to choose between two possible outcomes (such as Yes/No, Up/Down or Red/Green). Alternatively, you may want to perform multi-way classification

("multi-class"), in which the predictor is picking one of several possible outcomes (such as Small/Medium/Large or Spring/Summer/Autumn/Winter)? Daimensions™ can deal with any number of categories as long as there are enough samples per category in the training data.

Whether you go with binary or multi-class, you'll need to define your outcome categories clearly and make sure your training data is labelled accordingly. If labeling is hard because the task is difficult (as in the case of emotion classification), consider using more than one annotator to label your training data and explicitly measure agreement between the annotators to get consistency. Be aware that your predictor can never be better than your annotation. There is no clearer situation than this in which the notion of "garbage in, garbage out" applies!

Another thing to consider when creating your training data set is the balance between the number of training examples available for each output class. Ideally there should be an even balance of available training data across all outputs (targets).

Deployment context

It's also important to think up front about how your classification results will be used in your deployment workflow. This can vary from "just helping the humans out a little" to "fully automated decision-making".

You might, for instance:

- Deploy a predictor that helps a human by calling attention to or prioritizing something in an existing workflow, with the goal of increasing efficiency or effectiveness
- Create a predictor with the goal of fully replacing humans who are already doing a task, with the goal of doing things more reliably or at lower cost
- Invent a new predictor to do a task that humans were never previously able to do in order to expand production capabilities

As you progress through the steps laid out below, you'll be making choices that trade off various aspects of your predictor's performance. Having thought about how the predictor will actually be used before you have to make those choices will be very helpful.

Data preparation

Next comes the creation and gathering of your training data. A training data set consists of one or more columns of data that capture input variables used to build the predictor, combined with one column that contains the output (or "target"). The values of data in the output column have to match the output classes you chose above - and they have to match exactly. The input columns can be anything, as long as all the values in the column are of the same type.

Each row represents a single training data point. No matter which real-world situation or simulated model from which you're drawing your training data, what matters is that the input values in the row correspond reliably to the output value (target) in the row.

Choosing what to initially use as columns in your initial training data set usually comes down to a combination of what you think is likely to create good predictions and what can readily be gathered and used. The beauty of working with Daimensions™ is that once you've made an initial choice about what data to use, it's easy and quick to figure out whether your choice was a good one. The measurements done by Daimensions™ give you immediate feedback on what, if anything, needs to be done to improve your choices. And because Daimensions™ operates very quickly, doing a few iterations to achieve your training data plan is straightforward.

One important thing to keep in mind: A column that contains a unique value in each row (for example a database key) will never contribute to **generalization**. It is therefore advised to not include database keys or other unique ID columns in any table that is used for machine learning. *If you decide to include one or more columns of this sort in your training data set (perhaps for provenance or quality assurance reasons), you must be sure to tell Daimensions™ to ignore the column, using the -ignorecolumns option (see below).*

There are no minimum or maximum requirements for how many rows you include in your training data set. However, if you are doing multi-class prediction, we recommend that you start with at least 100 rows per output class.

Upload data

Data requirements

In order to run Daimensions™ on your data, we require the following:

- Labeled Data
- CSV Format (RFC4180)
- Either do not include an ID column or use the -ignorecolumns option when trainingrank
- For multi-class, provide a minimum of 100 data points per class

We also recommend the following practices:

- Putting headers in the first row of each column
- Placing the output column at the far right (otherwise you will have to explicitly tell the system which column is the output)

Important note: it is quite common to have one column be some kind of unique identifier for the row of data (typically alpha-numeric). If you do include this kind of column in your data, be sure to tell the system to *ignore this column*! We cover how to do this below.

Steps

1. Check that the Daimensions™ client is installed in the proper directory on your local machine:

```
Brainome-Pro Experiment % which btc
/usr/local/bin/btc
```

If it is not, please refer to the Daimensions™ CLI Quickstart for installation.

2. Verify that you can connect to and log in to the Daimensions™ cloud server by running a simple command:

```
Brainome-Pro Experiment % btc VERSION
Please login on daimensions.brainome.ai
Daimensions(tm) user email: user1@brainome.ai
Daimensions(tm) password: <enter password>
This is btc v0.97 (cloud).
Compatible with Daimensions(tm) versions v0.97 to v1.0.
```

3. Set up a directory to hold your training dataset:

```
Brainome-Pro Experiment % pwd
/Users/user1/Documents/Brainome/Experiment/Experiment_data

Brainome-Pro Experiment % ls -l
total 255560
-rw-r--r--@ 1 user1  staff  22115 Jul 26 14:07 experiment_test.csv
-rw-r--r--@ 1 user1  staff  22191 Jul 26 14:07 experiment_train.csv
```

Measure and evaluate

Why measure up front?

Every scientific endeavor starts with measurements, and so should machine learning. (Or, if you prefer, consider that just as in remodeling a kitchen or sewing a dress, being successful requires that you always measure before you cut.)

If you do the up-front work of measuring and "right-sizing" the training process before you build your machine learning model, you will save time and money by avoiding mistakes.

If you don't measure up front:

- You have no way of knowing whether the predictor you build will actually do what you want it to do when it sees new data that it wasn't trained on
- You will probably build a model that is much larger than it needs to be
- Your training times will probably be much longer than they need to be
- Your run times for the deployed predictor could be much longer than they need to be

- You can end up in a situation where you just don't know whether you have the right amount or right type of training data, even after extensive training and testing

Daimensions™ gives you the insight you need to be able to measure up front!

- **Learnability** (using capacity progression and risk)
- **Generalization** ratio
- **Noise resilience**
- **Attribute ranking**
- And, of course, all the standard **accuracy** figures

(For more information on **terminology in bold**, check out the **Daimensions™ Glossary**.)

Steps

1. Run the measuring process in your experiment directory

```
Brainome-Pro Experiment % pwd
/Users/user1/Documents/Brainome/Experiment

Brainome-Pro Experiment % ls -l Experiment_data
total 255560
-rw-r--r--@ 1 user1  staff  22115 Jul 26 14:07 experiment_test.csv
-rw-r--r--@ 1 user1  staff  22191 Jul 26 14:07 experiment_train.csv
Brainome-Pro Experiment % btc Experiment_data/experiment_train.csv -measureonly >
experiment.measurements
```

2. Look at the results

```
Brainome-Pro Experiment % pwd
/Users/user1/Documents/Brainome/Experiment

Brainome-Pro Experiment % ls
experiment.measurements

Brainome-Pro Experiment % more experiment.measurements
Brainome Daimensions(tm) 0.97 Copyright (c) 2019, 2020 by Brainome, Inc. All Rights
Reserved.
Licensed to: Brainome User 1
Expiration date: 2020-08-30 (35 days left)
Number of threads: 1
Maximum file size: 1GB
Connected to: https://brainome.ai:8080

Data:
Number of instances: 442
```

```
Number of attributes: 7
Number of classes: 2
Class balance: 60.63% 39.37%

Learnability:
Best guess accuracy: 60.63%
Capacity progression (# of decision points): [7, 8, 10, 10, 11, 11]
Decision Tree: 220 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 73 parameters

Risk that model needs to overfit for 100% accuracy...
using Decision Tree: 99.55%
using Neural Networks: 100.00%

Expected Generalization...
using Decision Tree: 2.01 bits/bit
using a Neural Network: 6.05 bits/bit

Recommendations:
Note: Maybe enough data to generalize. [yellow]
Warning: Data has high information density. Expect varying results and increase --effort.

Time estimate for a Neural Network:
Estimated time to architect: 0d 0h 0m 0s
Estimated time to prime (subject to change after model architecting): 0d 0h 3m 18s

Time estimate for Decision Tree:
Estimated time to prime a decision tree: a few seconds
```

Decide what to do next

Once you've taken and evaluated all these measurements, you're ready to decide what to do next.

To understand which way to go, you need to look at the measurement results you obtained from your training data.

If the **Learnability** "Recommendations" is green and your **Accuracy** and **Generalization** numbers look like they are where you'd like them to be, then you can move ahead to the next step, where you'll build your predictor.

However, in the example above, the **Learnability** "Recommendations" is yellow, not green. We're also seeing a warning that we may need to ask Daimensions™ to do some extra work (using the "-e" option, see below) to get the most out of what appears to be very information-dense data. So in this case, when we move on to build the predictor, we will already have some ideas about what we can do to get the best results.

Sometimes at this measurement-only stage Daimensions™ will give specific advice about which type of predictor to build (decision tree or neural network). In this case this did not

happen - there's nothing in the "-measureonly" output that gives specific feedback on this point. So our first approach will be to let Daimensions™ decide what kind of predictor it wants to build, and only get into trying to force one kind of model vs. another if needed later on.

Here are some common situations that you may encounter after running in "measure only" mode, together with suggested actions to take:

Measurements	Options/actions
Learnability "Recommendations" is red.	This means you don't have enough data or your data is not clean enough and Daimensions™ is having a hard time extracting a general predictor. You have the following options: 1) Use attribute selection to prioritize the most useful columns. Often tracking down the most important attributes first can help increase learnability by not using columns that are irrelevant. You can either ignore columns by using the **-ignorecolumns** option or use the automatic ranking algorithm via the **-rank** option*. *(*The **-rank** option is scheduled for release in Q4 2020.)* 2) If you're doing multi-class predictions, reduce the number of output classes. Sometimes ignoring classes can help learnability. Once you understand the problem better, it may be easier to select down irrelevant columns and add classes back in. 3) Re-check the quality and consistency of your training data annotation. What is the annotator agreement? Are the classes balanced? Higher accuracy in annotation combined with balanced classes will give better results. 4) If you are using perceptual data such as images, sound or videos, you may need to do some pre-processing. You may need to use a feature extractor or a convolution algorithm as a first step before using Daimensions™. 5) If you started with very few columns, add more attributes (columns) and try again. 6) If nothing else helps, try collecting more data of the same sort that you started with, then rerun your measurements.
Learnability "Recommendations" is yellow.	Choose the predictor (decision tree or neural network) with the highest generalization first. If that doesn't work well, increase **effort** with the **-e** training option. Try incrementally iincreasing effort levels to see if you can get better results. If you max out on available effort options, then review and try the options (in order!) listed above for "Learnability is

	red.”
Learnability “Recommendations” is green.	*Choose the predictor with the highest estimated generalization. The automatic choices that Daimensions™ makes should give you good results.*

It is rare that a data set will produce a “measure only” **Learnability** “Recommendations” result of “green” right out of the box. Because you’re unlikely to see this “green” outcome when working with your own data, we’re including a clear-cut example here for comparison. This is a simple dataset of 200 numbers in which all numbers <100 are class 0 and all numbers >= 100 are class 1:

```
Brainome-Pro Experiment % btc Experiment_data/simple_example_train.csv -measureonly >
simple_example.measurements

Brainome-Pro Experiment % ls
simple_example.measurements

Brainome-Pro Experiment % more simple_example.measurements
Brainome Daimensions(tm) 0.97 Copyright (c) 2019, 2020 by Brainome, Inc. All Rights
Reserved.
Licensed to: Brainome User 1
Expiration date: 2020-08-30 (35 days left)
Number of threads: 1
Maximum file size: 1GB
Connected to: https://brainome.ai:8080

Data:
Number of instances: 200
Number of attributes: 1
Number of classes: 2
Class balance: 49.0% 50.5%

Learnability:
Best guess accuracy: 50.50%
Capacity progression (# of decision points): [2, 2, 2, 2, 2, 2]

Decision Tree: 2 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 2 parameters
Risk that model needs to overfit for high accuracies...
using Decision Tree: 2.00%
using Neural Networks: 9.09%

Expected Generalization...
```

```
using Decision Tree: 100.00 bits/bit
using a Neural Network: 100.00 bits/bit

Recommendations:
Note: Enough data to generalize. [green]

Time estimate for Decision Tree:
Estimated time to prime a decision tree: a few seconds
Note: Assuming machine learner type DT for compiling efficiency (not accuracy).
Note: Choice can be overriden with -f parameter.

Classifier Type:                    Decision Tree
System Type:                        Binary classifier
Best-guess accuracy:                50.50%
Model accuracy:                     100.00% (200/200 correct)
Improvement over best guess:        49.50% (of possible 49.5%)
Model capacity (MEC):               1 bits
Generalization ratio:               200.00 bits/bit
Model efficiency:                   49.50%/parameter
System behavior
True Negatives:                     49.50% (99/200)
True Positives:                     50.50% (101/200)
False Negatives:                    0.00% (0/200)
False Positives:                    0.00% (0/200)
True Pos. Rate/Sensitivity/Recall:  1.00
True Neg. Rate/Specificity:         1.00
Precision:                          1.00
F-1 Measure:                        1.00
False Negative Rate/Miss Rate:      0.00
Critical Success Index:             1.00
```

It's not just simple data sets that can yield a "green" value for learnability Recommendations, however. Sometimes very large data sets can also yield a "green" result.

Most of the work you'll do with Daimensions™ will be with data sets that yield a "yellow" result. This is normal. This is by design. Use the measurement capabilities of Daimensions™ to understand what's going on in your data so that you can build a useful and reliable predictor while knowing exactly how much time, effort, and money to put into data collection. You don't need to dramatically over-collect data in an effort to get to a "green" outcome! Instead, it's much better to use all of the measurements provided by Daimensions™ together in concert so that you can figure out the right amount and type of data you need.

Rank and select attributes to use

Another very powerful thing to do before you build your model is to have Daimensions™ select which data attributes (columns) would be the best to use. To do that, use the **-rank** build option with **-measureonly**.

```
Brainome-Pro Experiment % btc titanic.csv -e 3 -f NN -rank -measureonly > titaniceNNrank.output
```

When we use **-rank** on our experiment data, we see that Daimensions™ chooses only to use a few of the attributes (columns) in the training data:

```
Brainome-Pro Experiment % more titaniceNNrank.output
...
Attribute Ranking:
Using only the important columns: Sex Fare Parents/Children Aboard Siblings/Spouses Aboard
.
.
.
```

Using the -rank option often gives valuable insight into the data itself, while saving considerable effort and cost before you move forward with model building.

Over time and working with different datasets, you will develop an intuition for what the measurements mean. Often the measurements work best when used as relative comparison between two choices. For example:

- If you have the choice between two training datasets, the one that achieves higher **generalization** over all possible machine learning algorithms is the one to use.
- If you are starting with a huge dataset, then you should definitely try to train on just a subset! You can choose that subset automatically by looking at how performance changes over the **capacity progression**, and use the smallest subset over which convergence arises.

Select build options

Daimensions™ has many control options that can be used to get the exact training and model behavior you want when you're building your predictor. Here are some of the training control options that many users find most helpful:

Control Option	Description	Example
-f	**Force** the creation of a specific type of machine learning model	**-f** NN forces the creation of a neural network

	(currently DT or NN). (The default behavior without the use of this option is to let Daimensions™ decide which kind of model to build.)	
-e	Tell Daimensions™ to make more **effort** than usual while training a model. (Default is e=1, max is 100).	-**e** 5 tells Daimensions™ to put 5 times more extra computation into its training efforts
-riskoverfit	Daimensions™ usually prioritizes generalization over accuracy. Tell Daimensions™ to reverse its default priority to make accuracy come first.	**-riskoverfit** tells Daimensions™ to prefer accuracy over generalization

Daimensions™ has other operational options that are useful when you're building a predictor, including:

Control Option	Description	Example
-v	Run Daimensions™ in **verbose** mode.	**-v** Daimensions™ sends a lot of status information to STDIO as it builds your predictor
-o	Write the predictor **into this file**.	**-o** mypredictor.py tells Daimensions™ to put your predictor into a file called "mypredictor.py"
-target	**Use a specific column** as the output/target.	**-target** Decision tells Daimensions™ to use the column whose header is "Decision" as the target/outcome column
-ignorecolumns	Tells Daimensions™ to **not use particular column(s)** as part of the training data. (This is typically	**-ignorecolumns** FileName,FileNo tells Daimensions™ to discard the columns labelled "FileName" and

	used for columns containing unique identifiers for rows/data points.)	"FileNo"
-rank [n]	Tells Daimensions™ to select only the most useful attributes (columns) in your data set when building a model. (This is typically used with data sets that have many attributes/columns.) You can specify a numeric argument to dictate how many columns to use.	**-rank** tells Daimensions™ to only use the most helpful attributes (columns) **-rank 5** forces Daimensions™ to pick and use the 5 most helpful attributes (columns)
-Wall	Tells Daimensions™ to display all **warnings**	**-Wall** sends any warnings generated by Daimensions™ to the standard output (typically your screen)

Build and run

Once you've considered which control and operational options you'd like to use, if any, you're ready to build your predictor!

Build predictor

After you've selected your options, you're ready to try your first build run.

Turning our attention back to our original example, here's how you ask Daimensions™ to build the predictor "experimente3.py", while spending some extra effort (in this case three times more effort "-e 3"), as was recommended above:

```
Brainome-Pro Experiment % btc -e 3 experiment_train.csv -o experiment_e3.py >
experiment_e3.buildnotes
```

Here's the Python predictor that DaimensionsTM creates:

```
Brainome-Pro Experiment % more experiment_e3.py
...
#!/usr/bin/env python3
#
# This code is was produced by an alpha version of Brainome Daimensions(tm) and is
```

```
# licensed under GNU GPL v2.0 or higher. For details, please see:
# https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html
#
#
# Output of Brainome Daimensions(tm) 0.97 Table Compiler v0.97.
# Invocation: btc Experiment_data/experiment_train.csv -e 3 -o experiment_e3.py
# Total compiler execution time: 0:07:56.19. Finished on: Jun-04-2020 21:40:31.
# This source code requires Python 3.
#
"""
Classifier Type:                    Decision Tree
System Type:                        Binary classifier
Best-guess accuracy:                61.44%
Model accuracy:                     81.51% (723/887 correct)
Improvement over best guess:        20.07% (of possible 38.56%)
Model capacity (MEC):               249 bits
Generalization ratio:               2.90 bits/bit
Model efficiency:                   0.08%/parameter
System behavior
True Negatives:                     52.54% (466/887)
True Positives:                     28.97% (257/887)
False Negatives:                    9.58% (85/887)
False Positives:                    8.91% (79/887)
True Pos. Rate/Sensitivity/Recall:  0.75
True Neg. Rate/Specificity:         0.86
Precision:                          0.76
F-1 Measure:                        0.76
False Negative Rate/Miss Rate:      0.25
Critical Success Index:             0.61
Overfitting:                        No
...
```

Here's another example on different data in which Daimensions™ is asked to build a neural network while being sure to ignore the column that contains a unique identifier for each training data point, and putting the resulting predictor in a file called "experimentNN.py":

```
Brainome-Pro Experiment % btc other_set_train.csv -f NN -ignorecolumns "dataID" -o
other_setNN.py > other_setNN.buildnotes
```

One thing that many users like to do is to track how long it takes Daimensions™ to build the predictor. You can find this information at the top of the predictor itself (in the Python file):

```
Brainome-Pro Experiment % more other_setNN.py
...
#!/usr/bin/env python3
#
# This code is was produced by an alpha version of Brainome Daimensions(tm) and is
```

```
# licensed under GNU GPL v2.0 or higher. For details, please see:
# https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html
#
#
# Output of Brainome Daimensions(tm) 0.96 Table Compiler v0.96.
# Invocation: btc Experiment_data/experiment_train.csv -f NN -ignorecolumns “dataID” -o
experimentNN.py
# Total compiler execution time: 0:07:56.19. Finished on: Jun-04-2020 21:40:31.
# This source code requires Python 3.
#
...
```

Another thing that you can do at this point is to take an initial look at your predictor's potential performance. Here's the information in the output that was generated when we ran the training on the second ("other") set:

```
Brainome-Pro Experiment % more other_setNN.buildnotes
…
Classifier Type: Neural Network
System Type:                         Binary classifier
Best-guess accuracy:                 57.36%
Model accuracy:                      58.32% (253723/435026 correct)
Improvement over best guess:         0.96% (of possible 42.64%)
Model capacity (MEC):                97 bits
Generalization ratio:                2615.70 bits/bit
Model efficiency:                    0.00%/parameter
System behavior
True Negatives:                      42.99% (187033/435026)
True Positives:                      15.33% (66690/435026)
```

Run predictor

Now it's time to try running your model on some new data, to check out your results.

The first thing to do is to make sure that Python3 is properly installed on your local machine:

```
Brainome-Pro Experiment % which python3
/Library/Frameworks/Python.framework/Versions/3.8/bin/python3
```

Now let's run our "three times the effort" "experiment" predictor from above. Let's run the predictor we built, "experiment_e3.py", on a new test data that Daimensions™ has not seen before:

```
Brainome-Pro Experiment % python3 experiment_e3.py Experiment_data/experiment.test.csv
-validate > experiment_e3.results
```

Evaluate results

Once your predictor has been run on new data, you can look at the output to get detailed measurements and explanations of its performance:

```
Brainome-Pro Experiment % more experiment_e3.results
…
Classifier Type:                    Decision Tree
System Type:                        Binary classifier
Best-guess accuracy:                62.92%
Model accuracy:                     73.03% (65/89 correct)
Improvement over best guess:        10.11% (of possible 37.08%)
Model capacity (MEC):               249 bits
Generalization ratio:               0.26 bits/bit
Model efficiency:                   0.04%/parameter
System behavior
True Negatives:                     51.69% (46/89)
True Positives:                     21.35% (19/89)
False Negatives:                    15.73% (14/89)
False Positives:                    11.24% (10/89)
True Pos. Rate/Sensitivity/Recall:  0.58
True Neg. Rate/Specificity:         0.82
Precision:                          0.66
F-1 Measure:                        0.61
False Negative Rate/Miss Rate:      0.42
Critical Success Index:             0.44
```

Once again you're at a decision point - is this predictor the one I want to deploy into my target workflow or not? Can I go with this or do I need to go back and make changes?

In our case, neither the results for Model **Accuracy** nor the **Generalization** Ratio are as strong as they might be. So we're going to go back and make a change. We're going to ask DaimensionsTM to build us a neural network instead of a decision tree

```
Brainome-Pro Experiment %  btc -e 3 experiment_train.csv -f NN -o experiment_e3NN.py >
experiment_e3NN.buildnotes
```

This change does create a good improvement in Model **Accuracy**, but more importantly, it greatly improves the **Generalization** Ratio. This is a good change to keep:

```
Brainome-Pro Experiment % more experiment_e3NN.results
…
Classifier Type:                    Neural Network
System Type:                        Binary classifier
Best-guess accuracy:                62.92%
```

```
Model accuracy:                     78.65% (70/89 correct)
Improvement over best guess:        15.73% (of possible 37.08%)
Model capacity (MEC):               19 bits
Generalization ratio:               3.68 bits/bit
Model efficiency:                   0.82%/parameter
System behavior
True Negatives:                     50.56% (45/89)
True Positives:                     28.09% (25/89)
False Negatives:                    8.99% (8/89)
False Positives:                    12.36% (11/89)
True Pos. Rate/Sensitivity/Recall:  0.76
True Neg. Rate/Specificity:         0.80
Precision:                          0.69
F-1 Measure:                        0.72
False Negative Rate/Miss Rate:      0.24
Critical Success Index:             0.57
```

In general, you want to choose the predictor with the highest **accuracy** and the highest **generalization**. If you don't see a warning that the predictor overfitted then it didn't. Unless you use "-riskoverfit", Daimensions™ always uses a held-out validation set. You can also use the bias meter to see if your predictor has bias built in.

If you're not satisfied with the **accuracy** or **generalization** of your predictor - don't worry! Here are some situations that you may encounter, together with suggested actions to take:

Situation	Options/actions
My predictor is near best guess accuracy or **accuracy** is just too low.	1) Check the measurements and the warnings for hints (see the section Decide What to Do Next above) 2) Use **-f** parameter to choose a different machine learner model 3) Use **-e** parameter to tell Daimensions™ to put more **effort** into training 4) Try using **-riskoverfit**. If the accuracy is still low, then the dataset is too noisy and needs either preprocessing or attribute selection (Decide What to Do Next)
I want higher **accuracy** at all cost.	Try **-riskoverfit**

My other predictor is 2% better...	Remember that Daimensions™ is an enterprise prediction system. Reliability in terms of reproducibility and resilience are goals that are just as important as accuracy. Data is always noisy and it's easy to overfit. Check the number of parameters of your other predictor and also run the Brainome Resilience Meter™ and the bias meter to compare all factors. This is what it takes to make a quality predictor that will perform well in the field.

If you're satisfied with the **accuracy** and **generalization** performance you're getting on new data, that's great! You can move on to deployment. Deploy the predictor just you would with any other piece of source code. The output predictor contains everything needed to run and validate predictions.

Deploy

Initial deployment

Daimensions™ produces code (typically Python3, but .exe will also be an option in future versions) that can be deployed anywhere, in any workflow or pipeline. It does not require any GPU support. Python3 predictors require the installation of numpy in addition to the standard Python3 libraries.

Inserting the output file produced by Daimensions™ into an existing production environment is by far the most common way that our users deploy their predictors. This approach usually fits seamlessly into existing build-QA-deploy-maintain operations cycles without major changes. This approach is so simple that, from an operations perspective, the predictor can usually be treated exactly like any other system component.

Ongoing deployment and reactions

If you have a stream of incoming data on which you are running a predictor and want to know if changes in operational conditions are changing your performance on it, you need to annotate the new data and retrain. Keep an eye out for the **memory equivalent capacity** measurements. It should not change for newly added data. If it does, it's highly recommended to reannotate and retrain.

Keep Going!

After completing these How-to Guides, you are good to go! Like everything else, mastery of measurements and model building can be achieved through practice. So we really suggest

you get going and have fun. We maintain a **Daimensions™ FAQ** where common questions are answered. You can also reach us via email at support@brainome.ai.