Titanic Using Daimensions

This notebook uses data from the Titanic competition on Kaggle (https://www.kaggle.com/c/titanic/overview).

Kaggle's description of the competition: "The sinking of the Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the widely considered 'unsinkable' RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren't enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others. In this challenge, we ask you to build a predictive model that answers the question: 'what sorts of people were more likely to survive?' using passenger data (ie name, age, gender, socio-economic class, etc)."

Goal: Make a predictor of survival from Titanic training data. We'll do this by using Daimensions to measure, build, and validate a predictor.

0. Getting Started

Because this is the very first tutorial, we'll go over how to install btc and get started. You can also see how to setup btc in the Daimensions Quickstart guide.

First, use the following link to download the installation script: https://download.brainome.net/btc-cli/btc-setup.sh. From the download directory, run the following bash command.

In [ ]:

```
! sh btc-setup.sh
```

The script will check that your operating system is supported, download the latest btc client to your machine and install it in /usr/local/bin. You will be prompted to enter the administrator password to install the software. *NOTE: After installation, make sure that "/usr/local/bin" is in your search path.*

Next, run the following command to wipe all cloud files. You will need your user credentials to login to DaimensionsTM. The first time you login, your license key will be downloaded automatically. Please use the default password that was provided to you.

In [ ]:

```
! btc WIPE
```

To change your password, use the following bash command.

In [ ]:

```
! btc CHPASSWD
```

1. Get Measurements

Measuring our data before building a predictor is important in order to avoid mistakes and optimize our model. If we don't measure our data, we have no way of knowing whether the predictor we build will actually do what we want it to do when it sees new data that it wasn't trained on. We'll probably build a model that is much larger than it needs to be, meaning our training and run times will probably be much longer than they need to be. We could end up in a situation where we just don't know whether we have the right amount or right type of training data, even after extensive training and testing. Because of these reasons, it's best to measure our data beforehand. Not to mention, Daimensions will tell us about learnability, the generalization ratio, noise resilience, and all the standard accuracy and confusion figures. For more information, you can read the Daimensions How-to Guide and Glossary.

In [1]:

```
# Below is a clip of the training data:
! head titanic_train.csv
# For Windows command prompt:
# type titanic_train.csv | more
```

As you can see from above, the target column (Survived) isn't the last column on the right. Because of this, we need to use '-target' so that Daimensions is looking at the correct target column for measuring and building a predictor.

In [2]:

```
# Measuring the training data:
! ./btc -measureonly titanic_train.csv -target Survived
```

```
Brainome Daimensions(tm) 0.99 Copyright (c) 2019, 2020 by Brainome, Inc. All Rights Reserved.
Licensed to: Alexander Makhratchev
Expiration date: 2021-04-30 (65 days left)
Number of threads: 1
Maximum file size: 30720MB
Running locally.
WARNING: Could not detect a GPU. Neural Network generation will be slow.

Data:
Number of instances: 891
Number of attributes: 11
Number of classes: 2
Class balance: 61.62% 38.38%

Learnability:
Best guess accuracy: 61.62%
Capacity progression: [8, 9, 10, 10, 11, 11]
Decision Tree: 419 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 118 parameters

Risk that model needs to overfit for 100% accuracy...
using Decision Tree: 99.42%
using Neural Networks: 100.00%

Expected Generalization...
using Decision Tree: 2.04 bits/bit
using a Neural Network: 7.25 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 2s

Estimated time to prime (subject to change after model architecting): 0d 0h 2m 53s

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

2. Build the Predictor

Because the learnability of the data (based on capacity progression and risk) is yellow, the how-to guide recommends to choose predictor with higher generalization and increase effort for best results. This means

using a neural network with effort should work best. Here, I'm using '-f NN' to make the predictor a neural network. I'm also using '-o predict.py' to output the predictor as a python file. To increase the effort, I'm using '-e 10' for 10 times the effort. Again, we have to use '-target Survived' because the target column isn't the last one.

In [3]:

```
# Building the predictor and outputting it to 'titanic_predict.py':
! ./btc -v -v -f NN titanic_train.csv -o titanic_predict.py -target Survived --yes
```

```
Brainome Daimensions(tm) 0.99 Copyright (c) 2019, 2020 by Brainome, Inc. All Rights Reser
ved.
Licensed to: Alexander Makhratchev
Expiration date: 2021-04-30 (65 days left)
Number of threads: 1
Maximum file size: 30720MB
Running locally.
WARNING: Could not detect a GPU. Neural Network generation will be slow.

Input: titanic_train.csv




Data:
Number of instances: 891
Number of attributes: 11
Number of classes: 2
Class balance: 61.62% 38.38%

Learnability:
Best guess accuracy: 61.62%
Capacity progression: [8, 9, 10, 10, 11, 11]
Decision Tree: 419 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 118 parameters

Risk that model needs to overfit for 100% accuracy...
using Decision Tree: 99.42%
using Neural Networks: 100.00%

Expected Generalization...
using Decision Tree: 2.04 bits/bit
using a Neural Network: 7.25 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 1s

Estimated time to prime (subject to change after model architecting): 0d 0h 3m 24s

Note: Machine learner type NN given by user.

Model capacity (MEC):     27 bits
Architecture efficiency:   1.0 bits/parameter


Estimated time to prime model: 0d 0h 3m 26s


Model created:
Sequential(
  (0): Linear(in_features=11, out_features=2, bias=True)
  (1): ReLU()
  (2): Linear(in_features=2, out_features=1, bias=True)
)



Classifier Type:                     Neural Network
System Type:                         Binary classifier
```

```
System Type:                         Binary classifier
Training/Validation Split:           60:40%
Best-guess accuracy:                 61.61%
Training accuracy:                   63.67% (340/534 correct)
Validation accuracy:                 58.54% (209/357 correct)
Overall Model accuracy:              61.61% (549/891 correct)
Overall Improvement over best guess: 0.00% (of possible 38.39%)
Model capacity (MEC):                27 bits
Generalization ratio:                19.53 bits/bit
Model efficiency:                    0.00%/parameter
System behavior
True Negatives:                      61.62% (549/891)
True Positives:                      0.00% (0/891)
False Negatives:                     38.38% (342/891)
False Positives:                     0.00% (0/891)
True Pos. Rate/Sensitivity/Recall:   0.00
True Neg. Rate/Specificity:          1.00
F-1 Measure:                         0.00
False Negative Rate/Miss Rate:       1.00
Critical Success Index:              0.00
Confusion Matrix:
 [61.62% 0.00%]
 [38.38% 0.00%]
Generalization efficiency:           9.62
Overfitting:                         No

Output: titanic_predict.py
READY.
```

3. Validate and Make Predictions

We've built our first predictor! Now it's time to put it to use. In the case of Titanic, we are given test data from Kaggle, where it's different from the training data and doesn't include 'Survival'. We can use the model we built to make predictions for the test data and submit it to Kaggle for its competition. In the following code, I'll save the model's prediction in 'titanic_prediction.csv'. You will see that the predictor appended the model's prediction of survival as the last column.

In [4]:

```
# Using predictor on test data and saving it to 'titanic_prediction.csv':
! python3 titanic_predict.py titanic_test.csv > titanic_prediction.csv
! head titanic_prediction.csv
```

```
PassengerId,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked,Prediction
892,3,"Kelly, Mr. James",male,34.5,0,0,330911,7.8292,,Q,0
893,3,"Wilkes, Mrs. James (Ellen Needs)",female,47,1,0,363272,7,,S,0
894,2,"Myles, Mr. Thomas Francis",male,62,0,0,240276,9.6875,,Q,0
895,3,"Wirz, Mr. Albert",male,27,0,0,315154,8.6625,,S,0
896,3,"Hirvonen, Mrs. Alexander (Helga E Lindqvist)",female,22,1,1,3101298,12.2875,,S,0
897,3,"Svensson, Mr. Johan Cervin",male,14,0,0,7538,9.225,,S,0
898,3,"Connolly, Miss. Kate",female,30,0,0,330972,7.6292,,Q,0
899,2,"Caldwell, Mr. Albert Francis",male,26,1,1,248738,29,,S,0
900,3,"Abrahim, Mrs. Joseph (Sophie Halaut Easu)",female,18,0,0,2657,7.2292,,C,0
```

If you have validation data, or data that has the target column but wasn't used for training, you can use it to validate the accuracy of your predictor, as we will do. For this particular instance, I found an annotated version of the Titanic test data, 'titanic_validation.csv', and used it to validate our model.

In [5]:

```
# To validate:
! python3 titanic_predict.py -validate titanic_validation.csv
```

```
Classifier Type:                    Neural Network
System Type:                        Binary classifier
Best-guess accuracy:                62.20%
Model accuracy:                     62.20% (260/418 correct)
Improvement over best guess:        0.00% (of possible 37.8%)
Model capacity (MEC):               27 bits
```

```
Generalization ratio:                 9.20 bits/bit
Model efficiency:                     0.00%/parameter
System behavior
True Negatives:                       62.20% (260/418)
True Positives:                       0.00% (0/418)
False Negatives:                      37.80% (158/418)
False Positives:                      0.00% (0/418)
True Pos. Rate/Sensitivity/Recall:    0.00
True Neg. Rate/Specificity:           1.00
F-1 Measure:                          0.00
False Negative Rate/Miss Rate:        1.00
Critical Success Index:               0.00
Confusion Matrix:
 [62.20% 0.00%]
 [37.80% 0.00%]
```

From validating the predictor, we can see that it has 74.64% accuracy, 12.44% better than best-guess accuracy (which classifies all data points as the majority class).

4. Improving Our Model

Our model did pretty well, but let's see if we can improve it. A column that contains a unique value in each row (for example a database key) will never contribute to generalization, so we shouldn't include database keys or other unique ID columns. We can remove these columns by using '-ignorecolumns'. We'll try ignoring columns: PassengerId, Name, Ticket, Cabin, Embarked, because they're all unique ID columns. We could also use '-rank' to rank columns by significance and only process contributing attributes.

Ignorecolumns vs Rank:

There may be situations where domain knowledge suggests a better choice of features than -rank. If we know the data generative process, we can do better with -ignorecolumns than with -rank. Rank is also optimizing for quick clustering/decision tree. For neural networks, we may still wish to reduce input features, which can be done with pca, but at the cost of interpretability. Some applications may require the original features are used in which case pca isn't viable. Ignorecolumns can reduce features while maintaining interpretability and work better for neural networks than -rank may, but the burden of choosing the right columns to keep is now on us.

Using -ignorecolumns:

In [6]:

```
# Using -ignorecolumns to make a better predictor:
! ./btc -v -v -f NN titanic_train.csv -o titanic_predict_igcol.py -target Survived -ignorecolumns PassengerId,Name,Ticket,Cabin,Embarked -e 10
```

```
Brainome Daimensions(tm) 0.99 Copyright (c) 2019, 2020 by Brainome, Inc. All Rights Reserved.
Licensed to: Alexander Makhratchev
Expiration date: 2021-04-30 (65 days left)
Number of threads: 1
Maximum file size: 30720MB
Running locally.
WARNING: Could not detect a GPU. Neural Network generation will be slow.

Input: titanic_train.csv




Data:
Number of instances: 891
Number of attributes: 6
Number of classes: 2
Class balance: 61.62% 38.38%

Learnability:
```

```
Best guess accuracy: 61.62%
Capacity progression: [9, 9, 10, 11, 11, 13]
Decision Tree: 224 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 73 parameters

Risk that model needs to overfit for 100% accuracy...
using Decision Tree: 55.32%
using Neural Networks: 100.00%

Expected Generalization...
using Decision Tree: 3.67 bits/bit
using a Neural Network: 11.73 bits/bit

Recommendations:
Warning: Not enough data to generalize. [red]
Time estimate for a Neural Network:
Estimated time to architect: 0d 0h 0m 2s

Estimated time to prime (subject to change after model architecting): 0d 0h 3m 26s

Note: Machine learner type NN given by user.

Model capacity (MEC):     17 bits
Architecture efficiency:   1.0 bits/parameter


Estimated time to prime model: 0d 0h 2m 57s



Estimated training time: 0d 0h 23m 8s


Model created:
Sequential(
  (0): Linear(in_features=6, out_features=2, bias=True)
  (1): ReLU()
  (2): Linear(in_features=2, out_features=1, bias=True)
)



Classifier Type:                    Neural Network
System Type:                        Binary classifier
Best-guess accuracy:                61.61%
Overall Model accuracy:             81.48% (726/891 correct)
Overall Improvement over best guess: 19.87% (of possible 38.39%)
Model capacity (MEC):               17 bits
Generalization ratio:               41.02 bits/bit
Model efficiency:                   1.16%/parameter
System behavior
True Negatives:                     57.91% (516/891)
True Positives:                     23.57% (210/891)
False Negatives:                    14.81% (132/891)
False Positives:                    3.70% (33/891)
True Pos. Rate/Sensitivity/Recall:  0.61
True Neg. Rate/Specificity:         0.94
Precision:                          0.86
F-1 Measure:                        0.72
False Negative Rate/Miss Rate:      0.39
Critical Success Index:             0.56
Confusion Matrix:
 [57.91% 3.70%]
 [14.81% 23.57%]
Generalization efficiency:          20.20
Overfitting:                        No
Note: Unable to split dataset. The predictor was trained and evaluated on the same data.

Output: titanic_predict_igcol.py
READY.
```

In [7]:

```
# Using the ignorecolumns predictor on test data and saving it to 'titanic_prediction_igc
ol.csv':
! python3 titanic_predict_igcol.py titanic_test.csv > titanic_prediction_igcol.csv
! head titanic_prediction_igcol.csv
```

```
PassengerId,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked,Prediction
892,3,"Kelly, Mr. James",male,34.5,0,0,330911,7.8292,,Q,0
893,3,"Wilkes, Mrs. James (Ellen Needs)",female,47,1,0,363272,7,,S,0
894,2,"Myles, Mr. Thomas Francis",male,62,0,0,240276,9.6875,,Q,0
895,3,"Wirz, Mr. Albert",male,27,0,0,315154,8.6625,,S,0
896,3,"Hirvonen, Mrs. Alexander (Helga E Lindqvist)",female,22,1,1,3101298,12.2875,,S,0
897,3,"Svensson, Mr. Johan Cervin",male,14,0,0,7538,9.225,,S,0
898,3,"Connolly, Miss. Kate",female,30,0,0,330972,7.6292,,Q,0
899,2,"Caldwell, Mr. Albert Francis",male,26,1,1,248738,29,,S,0
900,3,"Abrahim, Mrs. Joseph (Sophie Halaut Easu)",female,18,0,0,2657,7.2292,,C,1
```

As we wanted, -ignorecolumns removed the PassengerId, Name, Ticket, Cabin, and Embarked attributes. Next, we can use -validate to check the accuracy of our new predictor.

In [8]:

```
# Validating the -ignorecolumns predictor
! python3 titanic_predict_igcol.py -validate titanic_validation.csv
```

```
Classifier Type:                    Neural Network
System Type:                        Binary classifier
Best-guess accuracy:                62.20%
Model accuracy:                     76.55% (320/418 correct)
Improvement over best guess:        14.35% (of possible 37.8%)
Model capacity (MEC):               17 bits
Generalization ratio:               18.00 bits/bit
Model efficiency:                   0.84%/parameter
System behavior
True Negatives:                     54.78% (229/418)
True Positives:                     21.77% (91/418)
False Negatives:                    16.03% (67/418)
False Positives:                    7.42% (31/418)
True Pos. Rate/Sensitivity/Recall:  0.58
True Neg. Rate/Specificity:         0.88
Precision:                          0.75
F-1 Measure:                        0.65
False Negative Rate/Miss Rate:      0.42
Critical Success Index:             0.48
Confusion Matrix:
 [54.78% 7.42%]
 [16.03% 21.77%]
```

Using -ignorecolumns has improved our accuracy to 77.75% from 74.64% originally.

Using -rank:

In [9]:

```
# Using -rank to make a better predictor:
! ./btc -v -v -f NN titanic_train.csv -o titanic_predict_rank.py -target Survived -rank
--yes -e 10
```

```
Warning: Automatic ranking is not recommended for Neural Networks
Brainome Daimensions(tm) 0.99 Copyright (c) 2019, 2020 by Brainome, Inc. All Rights Reser
ved.
Licensed to: Alexander Makhratchev
Expiration date: 2021-04-30 (65 days left)
Number of threads: 1
Maximum file size: 30720MB
Running locally.
WARNING: Could not detect a GPU. Neural Network generation will be slow.

Input: titanic_train.csv
```

```
Attribute Ranking:
Using only the important columns: Sex SibSp Parch Pclass
Risk of coincidental column correlation: <0.001%

Data:
Number of instances: 891
Number of attributes: 4
Number of classes: 2
Class balance: 61.62% 38.38%

Learnability:
Best guess accuracy: 61.62%
Capacity progression: [8, 10, 11, 11, 13, 13]
Decision Tree: 1 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 49 parameters

Risk that model needs to overfit for 100% accuracy...
using Decision Tree: 0.29%
using Neural Networks: 89.09%

Expected Generalization...
using Decision Tree: 692.67 bits/bit
using a Neural Network: 17.47 bits/bit

Recommendations:
Note: Maybe enough data to generalize. [yellow]
Note: Decision Tree clustering may outperform Neural Networks. Try with -f DT.
Time estimate for a Neural Network:
Estimated time to architect: 0d 0h 0m 1s

Estimated time to prime (subject to change after model architecting): 0d 0h 2m 12s

Note: Machine learner type NN given by user.

Model capacity (MEC):     31 bits
Architecture efficiency:   1.0 bits/parameter


Estimated time to prime model: 0d 0h 2m 15s



Estimated training time: 0d 0h 18m 32s


Model created:
Sequential(
  (0): Linear(in_features=4, out_features=4, bias=True)
  (1): ReLU()
  (2): Linear(in_features=4, out_features=1, bias=True)
)



Classifier Type:                     Neural Network
System Type:                         Binary classifier
Training/Validation Split:           70:30%
Best-guess accuracy:                 61.61%
Training accuracy:                   83.30% (519/623 correct)
Validation accuracy:                 75.74% (203/268 correct)
Overall Model accuracy:              81.03% (722/891 correct)
Overall Improvement over best guess: 19.42% (of possible 38.39%)
Model capacity (MEC):                25 bits
Generalization ratio:                27.75 bits/bit
Model efficiency:                    0.77%/parameter
```

```
System behavior
True Negatives:                       55.22% (492/891)
True Positives:                       25.81% (230/891)
False Negatives:                      12.57% (112/891)
False Positives:                      6.40% (57/891)
True Pos. Rate/Sensitivity/Recall:    0.67
True Neg. Rate/Specificity:           0.90
Precision:                            0.80
F-1 Measure:                          0.73
False Negative Rate/Miss Rate:        0.33
Critical Success Index:               0.58
Confusion Matrix:
 [55.22% 6.40%]
 [12.57% 25.81%]
Generalization efficiency:            13.66
Overfitting:                          No
Using only the important columns: Sex SibSp Parch Pclass
Risk of coincidental column correlation: <0.001%


Output: titanic_predict_rank.py
READY.
```

In [10]:

```
# Using the rank predictor on test data and saving it to 'titanic_prediction_rank.csv':
! python3 titanic_predict_rank.py titanic_test.csv > titanic_prediction_rank.csv
! head titanic_prediction_rank.csv
```

```
PassengerId,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked,Prediction
892,3,"Kelly, Mr. James",male,34.5,0,0,330911,7.8292,,Q,0
893,3,"Wilkes, Mrs. James (Ellen Needs)",female,47,1,0,363272,7,,S,1
894,2,"Myles, Mr. Thomas Francis",male,62,0,0,240276,9.6875,,Q,0
895,3,"Wirz, Mr. Albert",male,27,0,0,315154,8.6625,,S,0
896,3,"Hirvonen, Mrs. Alexander (Helga E Lindqvist)",female,22,1,1,3101298,12.2875,,S,1
897,3,"Svensson, Mr. Johan Cervin",male,14,0,0,7538,9.225,,S,0
898,3,"Connolly, Miss. Kate",female,30,0,0,330972,7.6292,,Q,1
899,2,"Caldwell, Mr. Albert Francis",male,26,1,1,248738,29,,S,0
900,3,"Abrahim, Mrs. Joseph (Sophie Halaut Easu)",female,18,0,0,2657,7.2292,,C,1
```

You can see that -rank decided to only look at the columns 'Sex','Parch' (Parent/child), and 'Fare'. This makes a lot of sense that the determining factors for survival on the Titanic were sex, how many parents or children they had on board, and how much their fare was. Seeing what attributes -rank chooses gives us powerful insight into understanding our data and its correlations.

In [11]:

```
# Validating the -rank predictor
! python3 titanic_predict_rank.py -validate titanic_validation.csv
```

```
Classifier Type:                    Neural Network
System Type:                        Binary classifier
Best-guess accuracy:                62.20%
Model accuracy:                     77.75% (325/418 correct)
Improvement over best guess:        15.55% (of possible 37.8%)
Model capacity (MEC):               25 bits
Generalization ratio:               12.44 bits/bit
Model efficiency:                   0.62%/parameter
System behavior
True Negatives:                     52.39% (219/418)
True Positives:                     25.36% (106/418)
False Negatives:                    12.44% (52/418)
False Positives:                    9.81% (41/418)
True Pos. Rate/Sensitivity/Recall:  0.67
True Neg. Rate/Specificity:         0.84
Precision:                          0.72
F-1 Measure:                        0.70
False Negative Rate/Miss Rate:      0.33
Critical Success Index:             0.53
Confusion Matrix:
 [52.39% 9.81%]
```

```
 [12.44% 25.36%]
```

With -rank, our accuracy is 76.79%, again, an improvement over our original 74.64%.

5. Next Steps

Success! We've built our first predictor and used it to make predictions on the Titanic test data. From here, we can use our model on any new Titanic data or use other control options to try to improve our results even more. To check out some of the other control options, use '-h' to see the full list. You can also check out Brainome's How-to Guide and Glossary for more information.

In [12]:

```
! ./btc -h
```

```
usage: btc [-h] [-o [OUTPUT]] [-headerless] [-cm CLASSMAPPING] [-nc NCLASSES]
           [-l LANGUAGE] [-target TARGET] [-nsamples NSAMPLES]
           [-ignorecolumns IGNORECOLUMNS] [-ignorelabels IGNORELABELS]
           [-rank [ATTRIBUTERANK]] [-v] [--quiet] [-biasmeter] [-measureonly]
           [-json [JSON]] [-Wall] [-pedantic] [-nofun] [-f FORCEMODEL]
           [-e EFFORT] [--yes] [-stopat STOPAT] [-modelonly] [-riskoverfit]
           [-nopriming] [-novalidation] [-balance] [--runlocalonly]
           input [input ...]

Brainome Daimensions(tm) Table Compiler

positional arguments:
  input                 Table as CSV files and/or URLs.
                        Alternatively, one of: {VERSION, TERMINATE, WIPE, CHPASSWD, LOGO
UT}
                        VERSION: Return version and exit.
                        TERMINATE: Terminate all cloud processes.
                        WIPE: Delete all files in the cloud.
                        CHPASSWD: Change password.
                        LOGOUT: Force relogin.

optional arguments:
  -h, --help            show this help message and exit
  -o [OUTPUT], --output [OUTPUT]
                        Output predictor filename.
  -headerless           Headerless inputfile.
  -cm CLASSMAPPING, --classmapping CLASSMAPPING
                        Manually map class labels to contiguous numeric range. Json form
at. Example: {\"F\":0,\"T\":1}
  -nc NCLASSES, --nclasses NCLASSES
                        Specify number of classes. Stop if not matched by input.
  -l LANGUAGE, --language LANGUAGE
                        Predictor language: py, exe
  -target TARGET        Specify target attribute (name or number).
  -nsamples NSAMPLES    Work on n random samples (0 full dataset, default: 1000000). Bala
ncing is not performed.
  -ignorecolumns IGNORECOLUMNS
                        Comma-separated list of attributes to ignore (names or numbers).
  -ignorelabels IGNORELABELS
                        Comma-separated list of rows of classes to ignore.
  -rank [ATTRIBUTERANK], --attributerank [ATTRIBUTERANK]
                        Rank columns by significance, only process contributing attribute
s. If optional parameter n is given, force the use top n attributes.
  -v, --verbosity       Verbosity (debug level).
  --quiet               Quiet operation.
  -biasmeter            Measure bias (only NN).
  -measureonly          Only output measurements, no compilation.
  -json [JSON]          Output all measurement data in JSON format to filename.
  -Wall                 Display all warnings
  -pedantic             Display all notes and warnings.
  -nofun                Stop compilation if there are warnings.
  -f FORCEMODEL, --forcemodel FORCEMODEL
                        Force model type: DT, NN
  -e EFFORT, --effort EFFORT
```

```
                         1=<effort<100. More careful model creation. Default: 1
  --yes                  No interaction. Default to yes for all questions.
  -stopat STOPAT         Stop when percentage goal has been reached. Default: 100
  -modelonly             Output model only in ONNX file format. No predictor.
  -riskoverfit           Prioritize validation accuracy over generalization. Default: Prio
ritize generalization over accuracy.
  -nopriming             Do not prime the model.
  -novalidation          Do not measure validation scores for created predictor.
  -balance               Treat classes as if they were balanced.
  --runlocalonly         Keep all data local and do not use cloud service (default).
```

In [ ]:

```

```