Titanic

October 14, 2020

1 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.

1.1 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.

[ ]:
```
! 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.

[ ]:
```
! btc WIPE
```

To change your password, use the following bash command.

[ ]:
```
! btc CHPASSWD
```

1.2 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.

[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.

[2]:
```
# Measuring the training data:
! btc -measureonly titanic_train.csv -target Survived
```

```
Brainome Daimensions(tm) 0.97 Copyright (c) 2019, 2020 by Brainome, Inc. All
Rights Reserved.
Licensed to: Ariana Park
Expiration date: 2020-11-30 (132 days left)
Number of threads: 1
Maximum file size: 4GB
Connected to: https://beta.brainome.ai:8080

Data:
Number of instances: 891
```

```
Number of attributes: 11
Number of classes: 2
Class balance: 61.5% 38.5%

Learnability:
Best guess accuracy: 61.50%
Capacity progression (# of decision points): [8, 9, 10, 10, 11, 11]
Decision Tree: 422 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 118 parameters

Risk that model needs to overfit for 100% accuracy…
using Decision Tree: 94.73%
using Neural Networks: 100.00%

Expected Generalization…
using Decision Tree: 2.11 bits/bit
using a Neural Network: 7.55 bits/bit

Recommendations:
Note: Maybe enough data to generalize. [yellow]
Warning: Cannot find numpy. The output predictor may not run on this machine.

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
58s

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

1.3 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.

[1]:
```
# 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) 0.97 Copyright (c) 2019, 2020 by Brainome, Inc. All
Rights Reserved.
Licensed to: Ariana Park
Expiration date: 2020-11-30 (111 days left)
Number of threads: 1
Maximum file size: 4GB
```

```
Connected to Brainome cloud.

Running btc will overwrite existing titanic_predict.py. OK? [y/N] yes
Input: titanic_train.csv
Sampling…done.
Preprocessing…done.
Cleaning…done.
Splitting into training and validation…done.
Pre-training measurements…done.
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 (# of decision points): [8, 9, 10, 10, 11, 11]
Decision Tree: 421 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 118 parameters

Risk that model needs to overfit for 100% accuracy…
using Decision Tree: 94.73%
using Neural Networks: 100.00%

Expected Generalization…
using Decision Tree: 2.11 bits/bit
using a Neural Network: 7.55 bits/bit

Recommendations:
Note: Maybe enough data to generalize. [yellow]
Warning: Cannot find numpy. The output predictor may not run on this machine.

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
18s
Note: Machine learner type NN given by user.
Architecting model…done.
Model capacity (MEC):     27 bits
Architecture efficiency:   1.0 bits/parameter

Estimating time to prime model…done.
Estimated time to prime model: 0d 0h 2m 23s

Priming model…done.
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)
)

Compiling predictor…done.
Validating predictor…done.
Classifier Type:                    Neural Network
System Type:                        Binary classifier
Best-guess accuracy:                61.61%
Model accuracy:                     81.03% (722/891 correct)
Improvement over best guess:        19.42% (of possible 38.39%)
Model capacity (MEC):               27 bits
Generalization ratio:               26.74 bits/bit
Model efficiency:                   0.71%/parameter
System behavior
True Negatives:                     54.77% (488/891)
True Positives:                     26.26% (234/891)
False Negatives:                    12.12% (108/891)
False Positives:                    6.85% (61/891)
True Pos. Rate/Sensitivity/Recall:  0.68
True Neg. Rate/Specificity:         0.89
Precision:                          0.79
F-1 Measure:                        0.73
False Negative Rate/Miss Rate:      0.32
Critical Success Index:             0.58
Overfitting:                        No

Output: titanic_predict.py
READY.
```

1.4 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.

[1]:
```
# 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,Predictio
n
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.

[9]:
```
# 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:                     74.64% (312/418 correct)
Improvement over best guess:        12.44% (of possible 37.8%)
Model capacity (MEC):               27 bits
Generalization ratio:               11.55 bits/bit
Model efficiency:                   0.46%/parameter
System behavior
True Negatives:                     54.31% (227/418)
True Positives:                     20.33% (85/418)
False Negatives:                    17.46% (73/418)
False Positives:                    7.89% (33/418)
True Pos. Rate/Sensitivity/Recall:  0.54
True Neg. Rate/Specificity:         0.87
Precision:                          0.72
F-1 Measure:                        0.62
False Negative Rate/Miss Rate:      0.46
Critical Success Index:             0.45
```

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).

1.5 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.

1.5.1 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.

1.5.2 Using -ignorecolumns:

[5]:
```
# 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) 0.97 Copyright (c) 2019, 2020 by Brainome, Inc. All
Rights Reserved.
Licensed to: Ariana Park
Expiration date: 2020-11-30 (132 days left)
Number of threads: 1
Maximum file size: 4GB
Connected to: https://beta.brainome.ai:8080

Input: titanic_train.csv
Sampling…done.
Preprocessing…done.
Cleaning…done.
Splitting into training and validation…done.
Pre-training measurements…done.
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 (# of decision points): [7, 9, 10, 11, 11, 13]
Decision Tree: 285 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 73 parameters

Risk that model needs to overfit for 100% accuracy…
using Decision Tree: 63.97%
using Neural Networks: 100.00%

Expected Generalization…
using Decision Tree: 3.13 bits/bit
```

```
using a Neural Network: 12.21 bits/bit

Recommendations:
Warning: Not enough data to generalize. [red]
Warning: Cannot find numpy. The output predictor may not run on this machine.

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
12s
Note: Machine learner type NN given by user.
Architecting model…done.
Model capacity (MEC):     17 bits
Architecture efficiency:   1.0 bits/parameter

Estimating time to prime model…done.
Estimated time to prime model: 0d 0h 2m 19s

Priming model…done.
Estimating training time…done.
Estimated training time: 0d 0h 15m 31s

Training…done.
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)
)

Compiling predictor…done.
Validating predictor…done.
Classifier Type:                    Neural Network
System Type:                        Binary classifier
Best-guess accuracy:                61.61%
Model accuracy:                     81.48% (726/891 correct)
Improvement over best guess:        19.87% (of possible 38.39%)
Model capacity (MEC):               17 bits
Generalization ratio:               42.70 bits/bit
Model efficiency:                   1.16%/parameter
System behavior
True Negatives:                     58.02% (517/891)
True Positives:                     23.46% (209/891)
False Negatives:                    14.93% (133/891)
False Positives:                    3.59% (32/891)
True Pos. Rate/Sensitivity/Recall:  0.61
True Neg. Rate/Specificity:         0.94
Precision:                          0.87
```

```
F-1 Measure:                          0.72
False Negative Rate/Miss Rate:        0.39
Critical Success Index:               0.56
Overfitting:                          No

Output: titanic_predict_igcol.py
READY.
```

[4]:
```
# Using the ignorecolumns predictor on test data and saving it to␣
↪'titanic_prediction_igcol.csv':
! python3 titanic_predict_igcol.py titanic_test.csv > titanic_prediction_igcol.
↪csv
! head titanic_prediction_igcol.csv
```

```
Pclass,Sex,Age,SibSp,Parch,Fare,Prediction
3,male,34.5,0,0,7.8292,0
3,female,47,1,0,7,0
2,male,62,0,0,9.6875,0
3,male,27,0,0,8.6625,0
3,female,22,1,1,12.2875,0
3,male,14,0,0,9.225,0
3,female,30,0,0,7.6292,1
2,male,26,1,1,29,0
3,female,18,0,0,7.2292,1
```

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.

[2]:
```
# 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:                       77.75% (325/418 correct)
Improvement over best guess:          15.55% (of possible 37.8%)
Model capacity (MEC):                 17 bits
Generalization ratio:                 19.11 bits/bit
Model efficiency:                     0.91%/parameter
System behavior
True Negatives:                       56.22% (235/418)
True Positives:                       21.53% (90/418)
False Negatives:                      16.27% (68/418)
False Positives:                      5.98% (25/418)
True Pos. Rate/Sensitivity/Recall:    0.57
True Neg. Rate/Specificity:           0.90
Precision:                            0.78
F-1 Measure:                          0.66
```

```
False Negative Rate/Miss Rate:      0.43
Critical Success Index:             0.49
```

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

1.5.3 Using -rank:

[6]:
```
# 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
```

```
Brainome Daimensions(tm) 0.97 Copyright (c) 2019, 2020 by Brainome, Inc. All
Rights Reserved.
Licensed to: Ariana Park
Expiration date: 2020-11-30 (132 days left)
Number of threads: 1
Maximum file size: 4GB
Connected to: https://beta.brainome.ai:8080

Input: titanic_train.csv
Sampling…done.
Preprocessing…done.
Cleaning…done.
Ranking attributes…done.
Splitting into training and validation…done.
Pre-training measurements…done.

Attribute Ranking:
Using only the important columns: Sex SibSp Parch Pclass

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 (# of decision points): [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: 0.28%
using Neural Networks: 89.09%

Expected Generalization…
using Decision Tree: 721.00 bits/bit
```

```
using a Neural Network: 18.18 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 1m
25s
Note: Machine learner type NN given by user.
Architecting model…done.
Model capacity (MEC):     31 bits
Architecture efficiency:   1.0 bits/parameter

Estimating time to prime model…done.
Estimated time to prime model: 0d 0h 1m 25s

Priming model…done.
Estimating training time…done.
Estimated training time: 0d 0h 12m 23s

Training…done.
Model created:
Sequential(
  (0): Linear(in_features=4, out_features=8, bias=True)
  (1): ReLU()
  (2): Linear(in_features=8, out_features=1, bias=True)
)

Compiling predictor…done.
Validating predictor…done.
Classifier Type:                      Neural Network
System Type:                          Binary classifier
Training/Validation Split:            60:40%
Best-guess accuracy:                  61.61%
Overall Model accuracy:               81.03% (722/891 correct)
Overall Improvement over best guess: 19.42% (of possible 38.39%)
Model capacity (MEC):                 49 bits
Generalization ratio:                 14.73 bits/bit
Model efficiency:                     0.39%/parameter
System behavior
True Negatives:                       54.88% (489/891)
True Positives:                       26.15% (233/891)
False Negatives:                      12.23% (109/891)
False Positives:                      6.73% (60/891)
True Pos. Rate/Sensitivity/Recall:    0.68
True Neg. Rate/Specificity:           0.89
```

```
Precision:                          0.80
F-1 Measure:                        0.73
False Negative Rate/Miss Rate:      0.32
Critical Success Index:             0.58
Confusion Matrix:
 [54.88% 6.73%]
 [12.23% 26.15%]
Overfitting:                        No

name 'getLicense' is not defined
Output: titanic_predict_rank.py
READY.
```

[7]:
```
# 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
```

```
Pclass,Sex,SibSp,Parch,Prediction
3,male,0,0,0
3,female,1,0,1
2,male,0,0,0
3,male,0,0,0
3,female,1,1,1
3,male,0,0,0
3,female,0,0,1
2,male,1,1,0
3,female,0,0,1
```

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.

[8]:
```
# 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:                     77.27% (323/418 correct)
Improvement over best guess:        15.07% (of possible 37.8%)
Model capacity (MEC):               49 bits
Generalization ratio:               6.59 bits/bit
Model efficiency:                   0.30%/parameter
System behavior
True Negatives:                     52.15% (218/418)
True Positives:                     25.12% (105/418)
```

```
False Negatives:                    12.68% (53/418)
False Positives:                    10.05% (42/418)
True Pos. Rate/Sensitivity/Recall:  0.66
True Neg. Rate/Specificity:         0.84
Precision:                          0.71
F-1 Measure:                        0.69
False Negative Rate/Miss Rate:      0.34
Critical Success Index:             0.53
Confusion Matrix:
 [52.15% 10.05%]
 [12.68% 25.12%]
```

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

1.6 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.

[1]:
```
! 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]
           [-Wall] [-pedantic] [-nofun] [-f FORCEMODEL] [-O OPTIMIZE]
           [-e EFFORT] [--yes] [-stopat STOPAT] [-modelonly] [-riskoverfit]
           [-nopriming] [-novalidation]
           input [input ...]

Brainome Daimensions(tm) Table Compiler

positional arguments:
  input                 Table as CSV files and/or URLs.
                        Alternatively, one of: {VERSION, TERMINATE, WIPE,
CHPASSWD}
                        VERSION: return version and exit.
                        TERMINATE: terminate all cloud processes.
                        WIPE: deletes all files in the cloud.
                        CHPASSWD: Change password

optional arguments:
  -h, --help            show this help message and exit
  -o [OUTPUT], --output [OUTPUT]
                        Output predictor
  -headerless           Headerless inputfile
  -cm CLASSMAPPING, --classmapping CLASSMAPPING
```

```
                        Manually map class labels to contiguous numeric range.
Python dictionary format.
  -nc NCLASSES, --nclasses NCLASSES
                        Specify number of classes. Stop if not matched by input.
  -l LANGUAGE, --language LANGUAGE
                        Predictor language: py
  -target TARGET        Specify target attribute (name or number)
  -nsamples NSAMPLES    Work on n random samples (0 full dataset, default:
1000000). Balancing 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
attributes. 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.
  -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, RF, GMM
  -O OPTIMIZE, --optimize OPTIMIZE
                        Optimize for: accuracy, TP, F1
  -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: Prioritize generalization over accuracy.
  -nopriming            Do not prime the model
  -novalidation         Do not measure validation scores for created predictor.
```