**Modeling Airline Delay**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Modeling-Airline-Delay)

**Overview**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Overview)

Statistics on whether a flight was delayed and for how long are available from government databases for all the major carriers. It would be useful to be able to predict before scheduling a flight whether or not it was likely to be delayed. In this example, DataRobot will try to model whether a flight will be delayed, based on information such as the scheduled departure time and whether rained the day of the flight.

**Set Up**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Set-Up)

This example assumes that the DataRobot Python client package has been installed and configured with the credentials of a DataRobot user with API access permissions.

**Data Sources**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Data-Sources)

Information on flights and flight delays is made available by the Bureau of Transportation Statistics at <http://apps.bts.gov/xml/ontimesummarystatistics/src/dstat/OntimeSummaryDepatures.xml>. To narrow down the amount of data involved, the datasets assembled for this use case are limited to US Airways flights out of Boston Logan in 2013 and 2014, although the script for interacting with DataRobot is sufficiently general that any dataset with the correct format could be used. A flight was declared to be delayed if it ultimately took off at least fifteen minutes after its scheduled departure time.

In additional to flight information, each record in the prepared dataset notes the amount of rain and whether it rained on the day of the flight. This information came from the National Oceanic and Atmospheric Administration’s Quality Controlled Local Climatological Data, available at <http://www.ncdc.noaa.gov/qclcd/QCLCD>. By looking at the recorded daily summaries of the water equivalent precipitation at the Bostan Logan station, the daily rainfall for each day in 2013 and 2014 was measured. For some days, the QCLCD reports trace amounts of rainfall, which was recorded as 0 inches of rain.

**Dataset Structure**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Dataset-Structure)

Each row in the assembled dataset contains the following columns

* was\_delayed
  + boolean
  + whether the flight was delayed
* daily\_rainfall
  + float
  + the amount of rain, in inches, on the day of the flight
* did\_rain
  + bool
  + whether it rained on the day of the flight
* Carrier Code
  + str
  + the carrier code of the airline - US for all entries in assembled dataset
* Date
  + str (MM/DD/YYYY format)
  + the date of the flight
* Flight Number
  + str
  + the flight number for the flight
* Tail Number
  + str
  + the tail number of the aircraft
* Destination Airport
  + str
  + the three-letter airport code of the destination airport
* Scheduled Deparature Time
  + str
  + the 24-hour scheduled departure time of the flight, in the origin airport’s timezone

In [1]:

**import** pandas **as** pd

**import** datarobot **as** dr

In [2]:

data\_path **=** "logan-US-2013.csv"

logan\_2013 **=** pd**.**read\_csv(data\_path)

logan\_2013**.**head()

Out[2]:

|  | **was\_delayed** | **daily\_rainfall** | **did\_rain** | **Carrier Code** | **Date (MM/DD/YYYY)** | **Flight Number** | **Tail Number** | **Destination Airport** | **Scheduled Departure Time** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | False | 0 | False | US | 02/01/2013 | 225 | N662AW | PHX | 16:20 |
| **1** | False | 0 | False | US | 02/01/2013 | 280 | N822AW | PHX | 06:00 |
| **2** | False | 0 | False | US | 02/01/2013 | 303 | N653AW | CLT | 09:35 |
| **3** | True | 0 | False | US | 02/01/2013 | 604 | N640AW | PHX | 09:55 |
| **4** | False | 0 | False | US | 02/01/2013 | 722 | N715UW | PHL | 18:30 |

We want to be able to make predictions for future data, so the “date” column should be transformed in a way that avoids values that won’t be populated for future data:

In [3]:

**def** **prepare\_modeling\_dataset**(df):

date\_column\_name **=** 'Date (MM/DD/YYYY)'

date **=** pd**.**to\_datetime(df[date\_column\_name])

modeling\_df **=** df**.**drop(date\_column\_name, axis**=**1)

days **=** {0: 'Mon', 1: 'Tues', 2: 'Weds', 3: 'Thurs', 4: 'Fri', 5: 'Sat',

6: 'Sun'}

modeling\_df['day\_of\_week'] **=** date**.**apply(**lambda** x: days[x**.**dayofweek])

modeling\_df['month'] **=** date**.**dt**.**month

**return** modeling\_df

In [4]:

logan\_2013\_modeling **=** prepare\_modeling\_dataset(logan\_2013)

logan\_2013\_modeling**.**head()

Out[4]:

|  | **was\_delayed** | **daily\_rainfall** | **did\_rain** | **Carrier Code** | **Flight Number** | **Tail Number** | **Destination Airport** | **Scheduled Departure Time** | **day\_of\_week** | **month** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | False | 0 | False | US | 225 | N662AW | PHX | 16:20 | Fri | 2 |
| **1** | False | 0 | False | US | 280 | N822AW | PHX | 06:00 | Fri | 2 |
| **2** | False | 0 | False | US | 303 | N653AW | CLT | 09:35 | Fri | 2 |
| **3** | True | 0 | False | US | 604 | N640AW | PHX | 09:55 | Fri | 2 |
| **4** | False | 0 | False | US | 722 | N715UW | PHL | 18:30 | Fri | 2 |

**DataRobot Modeling**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#DataRobot-Modeling)

As part of this use case, in model\_flight\_ontime.py, a DataRobot project will be created and used to run a variety of models against the assembled datasets. By default, DataRobot will run autopilot on the automatically generated Informative Features list, which excludes certain pathological features (like Carrier Code in this example, which is always the same value), and we will also create a custom feature list excluding the amount of rainfall on the day of the flight.

This notebook shows how to use the Python API client to create a project, create feature lists, train models with different sample percents and feature lists, and view the models that have been run. It will:

* create a project
* create a new feature list (no foreknowledge) excluding the rainfall features
* set the target to was\_delayed, and run DataRobot autopilot on the Informative Features list
* rerun autopilot on a new feature list
* make predictions on a new data set

**Starting a Project**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Starting-a-Project)

In [5]:

project **=** dr**.**Project**.**start(logan\_2013\_modeling,

project\_name**=**'Airline Delays - was\_delayed',

target**=**"was\_delayed")

project**.**id

Out[5]:

u'57b64eacc808916b01929a5e'

**Jobs and the Project Queue**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Jobs-and-the-Project-Queue)

You can view the project in your browser:

In [6]:

project**.**open\_leaderboard\_browser()

Out[6]:

True

In [7]:

*# Set worker count higher. This will fail if you don't have 10 workers.*

project**.**set\_worker\_count(10)

Out[7]:

Project(Airline Delays - was\_delayed)

In [8]:

project**.**pause\_autopilot()

Out[8]:

True

In [9]:

*# More jobs will go in the queue in each stage of autopilot.*

*# This gets the currently inprogress and queued jobs*

project**.**get\_model\_jobs()

Out[9]:

[ModelJob(Naive Bayes combiner classifier, status=inprogress),

ModelJob(Breiman and Cutler Random Forest Classifier, status=inprogress),

ModelJob(RuleFit Classifier, status=queue),

ModelJob(Regularized Logistic Regression (L2), status=queue),

ModelJob(Elastic-Net Classifier (L2 / Binomial Deviance), status=queue),

ModelJob(RandomForest Classifier (Gini), status=queue),

ModelJob(eXtreme Gradient Boosted Trees Classifier with Early Stopping, status=queue),

ModelJob(Nystroem Kernel SVM Classifier, status=queue),

ModelJob(Regularized Logistic Regression (L2), status=queue),

ModelJob(Elastic-Net Classifier (L2 / Binomial Deviance), status=queue),

ModelJob(Elastic-Net Classifier (L2 / Binomial Deviance) with Binned numeric features, status=queue),

ModelJob(RandomForest Classifier (Gini), status=inprogress),

ModelJob(Elastic-Net Classifier (mixing alpha=0.5 / Binomial Deviance), status=queue),

ModelJob(RandomForest Classifier (Gini), status=queue),

ModelJob(RandomForest Classifier (Entropy), status=queue),

ModelJob(ExtraTrees Classifier (Gini), status=queue),

ModelJob(Gradient Boosted Trees Classifier with Early Stopping, status=queue),

ModelJob(Gradient Boosted Greedy Trees Classifier with Early Stopping, status=queue),

ModelJob(eXtreme Gradient Boosted Trees Classifier with Early Stopping, status=queue),

ModelJob(Auto-tuned K-Nearest Neighbors Classifier (Minkowski Distance), status=queue),

ModelJob(Gradient Boosted Trees Classifier, status=inprogress),

ModelJob(Decision Tree Classifier (Gini), status=inprogress),

ModelJob(Logistic Regression, status=inprogress),

ModelJob(Regularized Logistic Regression (L2), status=inprogress),

ModelJob(Auto-tuned K-Nearest Neighbors Classifier (Minkowski Distance), status=inprogress),

ModelJob(Majority Class Classifier, status=inprogress),

ModelJob(Gradient Boosted Trees Classifier, status=inprogress)]

In [10]:

project**.**unpause\_autopilot()

Out[10]:

True

**Features**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Features)

In [11]:

features **=** project**.**get\_features()

features

Out[11]:

[Feature(did\_rain),

Feature(Destination Airport),

Feature(Carrier Code),

Feature(Flight Number),

Feature(Tail Number),

Feature(day\_of\_week),

Feature(month),

Feature(Scheduled Departure Time),

Feature(daily\_rainfall),

Feature(was\_delayed)]

In [12]:

pd**.**DataFrame([f**.**\_\_dict\_\_ **for** f **in** features])

Out[12]:

|  | **feature\_type** | **id** | **importance** | **low\_information** | **na\_count** | **name** | **unique\_count** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Boolean | 2 | 0.029045 | False | 0 | did\_rain | 2 |
| **1** | Categorical | 6 | 0.003714 | True | 0 | Destination Airport | 5 |
| **2** | Categorical | 3 | NaN | True | 0 | Carrier Code | 1 |
| **3** | Numeric | 4 | 0.005900 | False | 0 | Flight Number | 329 |
| **4** | Categorical | 5 | -0.004512 | True | 0 | Tail Number | 296 |
| **5** | Categorical | 8 | 0.003452 | True | 0 | day\_of\_week | 7 |
| **6** | Numeric | 9 | 0.003043 | True | 0 | month | 12 |
| **7** | Time | 7 | 0.058245 | False | 0 | Scheduled Departure Time | 77 |
| **8** | Numeric | 1 | 0.034295 | False | 0 | daily\_rainfall | 58 |
| **9** | Boolean | 0 | 1.000000 | False | 0 | was\_delayed | 2 |

Three feature lists are automatically created:

* *Raw Features*: one for all features
* *Informative Features*: one based on features with any information (columns with no variation are excluded)
* *Univariate Importance*: one based on univariate importance (this is only created after the target is set)

*Informative Features* is the one used by default in autopilot.

In [13]:

feature\_lists **=** project**.**get\_featurelists()

feature\_lists

Out[13]:

[Featurelist(Informative Features),

Featurelist(Raw Features),

Featurelist(Univariate Selections)]

In [14]:

*# create a featurelist without the rain features*

*# (since they leak future information)*

informative\_feats **=** [lst **for** lst **in** feature\_lists **if**

lst**.**name **==** 'Informative Features'][0]

no\_foreknowledge\_features **=** list(

set(informative\_feats**.**features) **-** {'daily\_rainfall', 'did\_rain'})

In [15]:

no\_foreknowledge **=** project**.**create\_featurelist('no foreknowledge',

no\_foreknowledge\_features)

no\_foreknowledge

Out[15]:

Featurelist(no foreknowledge)

In [16]:

project**.**get\_status()

Out[16]:

{u'autopilot\_done': False,

u'stage': u'modeling',

u'stage\_description': u'Ready for modeling'}

In [17]:

*# This waits until autopilot is complete:*

project**.**wait\_for\_autopilot(check\_interval**=**90)

In progress: 10, queued: 17 (waited: 0s)

In progress: 10, queued: 17 (waited: 1s)

In progress: 10, queued: 17 (waited: 1s)

In progress: 10, queued: 17 (waited: 2s)

In progress: 10, queued: 17 (waited: 3s)

In progress: 10, queued: 17 (waited: 4s)

In progress: 10, queued: 16 (waited: 8s)

In progress: 10, queued: 10 (waited: 14s)

In progress: 9, queued: 4 (waited: 27s)

In progress: 4, queued: 0 (waited: 53s)

In progress: 7, queued: 0 (waited: 105s)

In progress: 10, queued: 22 (waited: 195s)

In progress: 10, queued: 1 (waited: 285s)

In progress: 4, queued: 0 (waited: 376s)

In progress: 0, queued: 0 (waited: 466s)

In [18]:

project**.**start\_autopilot(no\_foreknowledge**.**id)

In [19]:

project**.**wait\_for\_autopilot(check\_interval**=**90)

In progress: 10, queued: 17 (waited: 0s)

In progress: 10, queued: 17 (waited: 0s)

In progress: 10, queued: 17 (waited: 1s)

In progress: 10, queued: 17 (waited: 1s)

In progress: 10, queued: 17 (waited: 2s)

In progress: 10, queued: 17 (waited: 4s)

In progress: 8, queued: 16 (waited: 8s)

In progress: 9, queued: 11 (waited: 14s)

In progress: 10, queued: 3 (waited: 27s)

In progress: 2, queued: 0 (waited: 53s)

In progress: 5, queued: 0 (waited: 104s)

In progress: 2, queued: 0 (waited: 195s)

In progress: 10, queued: 0 (waited: 285s)

In progress: 3, queued: 0 (waited: 376s)

In progress: 0, queued: 0 (waited: 467s)

**Models**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Models)

In [20]:

models **=** project**.**get\_models()

example\_model **=** models[0]

example\_model

Out[20]:

Model(ENET Blender)

Models represent fitted models and have various data about the model, including metrics:

In [21]:

example\_model**.**metrics

Out[21]:

{u'AUC': {u'crossValidation': 0.759074,

u'holdout': None,

u'validation': 0.75202},

u'FVE Binomial': {u'crossValidation': 0.145768,

u'holdout': None,

u'validation': 0.14552},

u'Gini Norm': {u'crossValidation': 0.5181479999999999,

u'holdout': None,

u'validation': 0.50404},

u'LogLoss': {u'crossValidation': 0.273184,

u'holdout': None,

u'validation': 0.27339},

u'RMSE': {u'crossValidation': 0.276848,

u'holdout': None,

u'validation': 0.27643},

u'Rate@Top10%': {u'crossValidation': 0.368602,

u'holdout': None,

u'validation': 0.37884},

u'Rate@Top5%': {u'crossValidation': 0.47755200000000003,

u'holdout': None,

u'validation': 0.4898},

u'Rate@TopTenth%': {u'crossValidation': 0.8,

u'holdout': None,

u'validation': 1.0}}

In [22]:

**def** **sorted\_by\_log\_loss**(models, test\_set):

models\_with\_score **=** [model **for** model **in** models **if**

model**.**metrics['LogLoss'][test\_set] **is** **not** None]

**return** sorted(models\_with\_score,

key**=lambda** model: model**.**metrics['LogLoss'][test\_set])

Let’s choose the models (from each feature set, to compare the scores) with the best LogLoss score from those with the rain and those without:

In [23]:

models **=** project**.**get\_models()

fair\_models **=** [mod **for** mod **in** models **if**

mod**.**featurelist\_id **==** no\_foreknowledge**.**id]

rain\_cheat\_models **=** [mod **for** mod **in** models **if**

mod**.**featurelist\_id **==** informative\_feats**.**id]

In [24]:

best\_fair\_model **=** sorted\_by\_log\_loss(fair\_models, 'crossValidation')[0]

best\_cheat\_model **=** sorted\_by\_log\_loss(rain\_cheat\_models, 'crossValidation')[0]

best\_fair\_model**.**metrics, best\_cheat\_model**.**metrics

Out[24]:

({u'AUC': {u'crossValidation': 0.7132059999999999,

u'holdout': None,

u'validation': 0.71313},

u'FVE Binomial': {u'crossValidation': 0.090004,

u'holdout': None,

u'validation': 0.08903},

u'Gini Norm': {u'crossValidation': 0.426412,

u'holdout': None,

u'validation': 0.42626},

u'LogLoss': {u'crossValidation': 0.29101999999999995,

u'holdout': None,

u'validation': 0.29147},

u'RMSE': {u'crossValidation': 0.286056,

u'holdout': None,

u'validation': 0.28639},

u'Rate@Top10%': {u'crossValidation': 0.29126799999999997,

u'holdout': None,

u'validation': 0.27645},

u'Rate@Top5%': {u'crossValidation': 0.367348,

u'holdout': None,

u'validation': 0.40816},

u'Rate@TopTenth%': {u'crossValidation': 0.6,

u'holdout': None,

u'validation': 0.66667}},

{u'AUC': {u'crossValidation': 0.759074,

u'holdout': None,

u'validation': 0.75202},

u'FVE Binomial': {u'crossValidation': 0.145768,

u'holdout': None,

u'validation': 0.14552},

u'Gini Norm': {u'crossValidation': 0.5181479999999999,

u'holdout': None,

u'validation': 0.50404},

u'LogLoss': {u'crossValidation': 0.273184,

u'holdout': None,

u'validation': 0.27339},

u'RMSE': {u'crossValidation': 0.276848,

u'holdout': None,

u'validation': 0.27643},

u'Rate@Top10%': {u'crossValidation': 0.368602,

u'holdout': None,

u'validation': 0.37884},

u'Rate@Top5%': {u'crossValidation': 0.47755200000000003,

u'holdout': None,

u'validation': 0.4898},

u'Rate@TopTenth%': {u'crossValidation': 0.8,

u'holdout': None,

u'validation': 1.0}})

**Visualizing Models**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Visualizing-Models)

This is a good time to use Model XRay (not yet available via the API) to visualize the models:

In [25]:

best\_fair\_model**.**open\_model\_browser()

Out[25]:

True

In [26]:

best\_cheat\_model**.**open\_model\_browser()

Out[26]:

True

**Unlocking the Holdout**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Unlocking-the-Holdout)

To maintain holdout scores as a valid estimate of out-of-sample error, we recommend not looking at them until late in the project. For this reason, holdout scores are locked until you unlock them.

In [27]:

project**.**unlock\_holdout()

Out[27]:

Project(Airline Delays - was\_delayed)

In [28]:

best\_fair\_model **=** dr**.**Model**.**get(project**.**id, best\_fair\_model**.**id)

best\_cheat\_model **=** dr**.**Model**.**get(project**.**id, best\_cheat\_model**.**id)

In [29]:

best\_fair\_model**.**metrics['LogLoss'], best\_cheat\_model**.**metrics['LogLoss']

Out[29]:

({u'crossValidation': 0.29101999999999995,

u'holdout': 0.29315,

u'validation': 0.29147},

{u'crossValidation': 0.273184, u'holdout': 0.27413, u'validation': 0.27339})

**Retrain on 100%**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Retrain-on-100%)

When ready to use the final model, you will probably get the best performance by retraining on 100% of the data.

In [30]:

model\_job\_fair\_100pct\_id **=** best\_fair\_model**.**train(sample\_pct**=**100)

Wait for the model to complete:

In [31]:

model\_fair\_100pct **=** dr**.**models**.**modeljob**.**wait\_for\_async\_model\_creation(

project**.**id, model\_job\_fair\_100pct\_id)

**Predictions**[**¶**](http://pythonhosted.org/datarobot/examples/airline_ontime_example/Modeling_Airline_Delay.html#Predictions)

Let’s make predictions for some new data. This new data will need to have the same transformations applied as we applied to the training data.

In [32]:

logan\_2014 **=** pd**.**read\_csv("logan-US-2014.csv")

logan\_2014\_modeling **=** prepare\_modeling\_dataset(logan\_2014)

logan\_2014\_modeling**.**head()

Out[32]:

|  | **was\_delayed** | **daily\_rainfall** | **did\_rain** | **Carrier Code** | **Flight Number** | **Tail Number** | **Destination Airport** | **Scheduled Departure Time** | **day\_of\_week** | **month** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | False | 0 | False | US | 450 | N809AW | PHX | 10:00 | Sat | 2 |
| **1** | False | 0 | False | US | 553 | N814AW | PHL | 07:00 | Sat | 2 |
| **2** | False | 0 | False | US | 582 | N820AW | PHX | 06:10 | Sat | 2 |
| **3** | False | 0 | False | US | 601 | N678AW | PHX | 16:20 | Sat | 2 |
| **4** | False | 0 | False | US | 657 | N662AW | CLT | 09:45 | Sat | 2 |

In [33]:

prediction\_dataset **=** project**.**upload\_dataset(logan\_2014\_modeling)

predict\_job **=** model\_fair\_100pct**.**request\_predictions(prediction\_dataset**.**id)

In [34]:

predictions **=** predict\_job**.**get\_result\_when\_complete()

In [35]:

pd**.**concat([logan\_2014\_modeling, predictions], axis**=**1)**.**head()

Out[35]:

|  | **was\_delayed** | **daily\_rainfall** | **did\_rain** | **Carrier Code** | **Flight Number** | **Tail Number** | **Destination Airport** | **Scheduled Departure Time** | **day\_of\_week** | **month** | **positive\_probability** | **prediction** | **row\_id** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | False | 0 | False | US | 450 | N809AW | PHX | 10:00 | Sat | 2 | 0.061519 | 0 | 0 |
| **1** | False | 0 | False | US | 553 | N814AW | PHL | 07:00 | Sat | 2 | 0.047499 | 0 | 1 |
| **2** | False | 0 | False | US | 582 | N820AW | PHX | 06:10 | Sat | 2 | 0.029104 | 0 | 2 |
| **3** | False | 0 | False | US | 601 | N678AW | PHX | 16:20 | Sat | 2 | 0.161229 | 0 | 3 |
| **4** | False | 0 | False | US | 657 | N662AW | CLT | 09:45 | Sat |  |  |  |  |