This tutorial was taken from an H2O tutorial online: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/starting-h2o.html

In [1]:

```
import h2o

from h2o.estimators.gbm import H2OGradientBoostingEstimator


h2o.init(max_mem_size=4)
```

```
Checking whether there is an H2O instance running at http://localhost:54321 ..... not
found.
Attempting to start a local H2O server...
; Java HotSpot(TM) 64-Bit Server VM (build 13.0.2+8, mixed mode, sharing)
  Starting server from c:\users\m\anaconda2\envs\python36\lib\site-packages\h2o\backen
d\bin\h2o.jar
  Ice root: C:\Users\m\AppData\Local\Temp\tmp8r46t_2b
  JVM stdout: C:\Users\m\AppData\Local\Temp\tmp8r46t_2b\h2o_m_started_from_python.out
  JVM stderr: C:\Users\m\AppData\Local\Temp\tmp8r46t_2b\h2o_m_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321 ... successful.
```

H2O cluster uptime:	09 secs
H2O cluster timezone:	America/Los_Angeles
H2O data parsing timezone:	UTC
H2O cluster version:	3.28.0.2
H2O cluster version age:	14 days, 20 hours and 35 minutes
H2O cluster name:	H2O_from_python_m_6gh42v
H2O cluster total nodes:	1
H2O cluster free memory:	4 Gb
H2O cluster total cores:	2
H2O cluster allowed cores:	2
H2O cluster status:	locked, healthy
H2O connection url:	http://127.0.0.1:54321
H2O connection proxy:	{'http': None, 'https': None}
H2O internal security:	False
H2O API Extensions:	Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4

Python version:	3.6.9 final

This data is from kaggle when googling 'airline data h2o' because the tutorial file was not a valid web page.

In [2]:

```
flights2 = h2o.import_file("flights.csv")
```

```
Parse progress: |█████████████████████████████████████████████████████████| 100%
```

In [3]:

```
flights2
flights2.shape
```

Out[3]:

```
(5819079, 31)
```

In [4]:

```
flights2.head()
```

YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	DEPARTURE_TIME
2015	1	1	4	AS	98	N407AS	ANC	SEA	5	2354
2015	1	1	4	AA	2336	N3KUAA	LAX	PBI	10	2
2015	1	1	4	US	840	N171US	SFO	CLT	20	18
2015	1	1	4	AA	258	N3HYAA	LAX	MIA	20	15
2015	1	1	4	AS	135	N527AS	SEA	ANC	25	24
2015	1	1	4	DL	806	N3730B	SFO	MSP	25	20
2015	1	1	4	NK	612	N635NK	LAS	MSP	25	19
2015	1	1	4	US	2013	N584UW	LAX	CLT	30	44
2015	1	1	4	AA	1112	N3LAAA	SFO	DFW	30	19
2015	1	1	4	DL	1173	N826DN	LAS	ATL	30	33

Out[4]:

In [5]:

```
flights2.columns
```

Out[5]:

```
['YEAR',
 'MONTH',
 'DAY',
 'DAY_OF_WEEK',
 'AIRLINE',
 'FLIGHT_NUMBER',
 'TAIL_NUMBER',
 'ORIGIN_AIRPORT',
 'DESTINATION_AIRPORT',
 'SCHEDULED_DEPARTURE',
 'DEPARTURE_TIME',
 'DEPARTURE_DELAY',
 'TAXI_OUT',
 'WHEELS_OFF',
 'SCHEDULED_TIME',
 'ELAPSED_TIME',
 'AIR_TIME',
 'DISTANCE',
 'WHEELS_ON',
 'TAXI_IN',
 'SCHEDULED_ARRIVAL',
 'ARRIVAL_TIME',
 'ARRIVAL_DELAY',
 'DIVERTED',
 'CANCELLED',
 'CANCELLATION_REASON',
 'AIR_SYSTEM_DELAY',
 'SECURITY_DELAY',
 'AIRLINE_DELAY',
 'LATE_AIRCRAFT_DELAY',
 'WEATHER_DELAY']
```

In [6]:

```
flights2['YEAR'] = flights2['YEAR'].asfactor()
flights2['MONTH'] = flights2['MONTH'].asfactor()
flights2['DAY_OF_WEEK'] = flights2['DAY_OF_WEEK'].asfactor()
flights2['FLIGHT_NUMBER'] = flights2['FLIGHT_NUMBER'].asfactor()
flights2['CANCELLED'] = flights2['CANCELLED'].asfactor()
#flights2['DEPARTURE_DELAY'] = flights2['DEPARTURE_DELAY'].asfactor()
```

In [7]:

```
predictors = ['YEAR', 'ORIGIN_AIRPORT','DESTINATION_AIRPORT','MONTH', 'DAY_OF_WEEK',
              'FLIGHT_NUMBER','DISTANCE','AIRLINE']
response = 'DEPARTURE_DELAY'
```

In [8]:

```
train, valid = flights2.split_frame(ratios=[0.8], seed=1234)
```

In [9]:

```
bin_num = [8,16,32,64,128,256,512,1024,2048,4096]
label = ["8","16","32","64","128","256","512","1024","2048","4096"]
```

The next command shows the attributes available in the H2OGradientBoostingEstimator function used to train the GBM model and test on the validation set with.

In [10]:

```
dir(H2OGradientBoostingEstimator)
```

Out[10]:

```
['_ModelBase__generate_partial_plots',
 '_ModelBase__generate_user_splits',
 '_ModelBase__grabValues',
 '_ModelBase__plot_1dpdp',
 '_ModelBase__plot_2dpdp',
 '_ModelBase__predFor3D',
 '_ModelBase__setAxs1D',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_additional_used_columns',
 '_bc',
 '_check_and_save_parm',
 '_check_targets',
 '_compute_algo',
 '_get_metrics',
 '_keyify_if_h2oframe',
 '_metrics_class',
 '_plot',
 '_print_model_scoring_history',
 '_requires_training_frame',
 '_resolve_model',
 '_train',
 '_verify_training_frame_params',
 'actual_params',
 'aic',
 'algo',
 'auc',
 'aucpr',
 'balance_classes',
 'biases',
 'build_tree_one_node',
 'calibrate_model',
 'calibration_frame',
 'categorical_encoding',
 'catoffsets',
 'check_constant_response',
 'checkpoint',
 'class_sampling_factors',
 'coef',
 'coef_norm',
 'col_sample_rate',
 'col_sample_rate_change_per_level',
 'col_sample_rate_per_tree',
 'convert_H2OXGBoostParams_2_XGBoostParams',
 'cross_validation_fold_assignment',
 'cross_validation_holdout_predictions',
 'cross_validation_metrics_summary',
 'cross_validation_models',
 'cross_validation_predictions',
 'custom_distribution_func',
 'custom_metric_func',
 'deepfeatures',
 'default_params',
 'detach',
 'distribution',
 'download_model',
 'download_mojo',
 'download_pojo',
 'end_time',
 'export_checkpoints_dir',
 'feature_frequencies',
 'fit',
 'fold_assignment',
 'fold_column',
 'full_parameters',
 'get_params',
 'get_xval_models',
 'gini',
 'have_mojo',
 'have_pojo',
 'histogram_type',
 'huber_alpha',
 'ignore_const_cols',
 'ignored_columns',
 'is_cross_validated',
 'join',
 'keep_cross_validation_fold_assignment',
 'keep_cross_validation_models',
 'keep_cross_validation_predictions',
 'key',
 'learn_rate',
 'learn_rate_annealing',
 'logloss',
 'mae',
 'max_abs_leafnode_pred',
 'max_after_balance_size',
 'max_confusion_matrix_size',
 'max_depth',
 'max_hit_ratio_k',
 'max_runtime_secs',
 'mean_residual_deviance',
 'min_rows',
 'min_split_improvement',
 'mixin',
 'model_id',
 'model_performance',
 'monotone_constraints',
 'mse',
 'nbins',
 'nbins_cats',
 'nbins_top_level',
 'nfolds',
 'normmul',
 'normsub',
 'ntrees',
 'ntrees_actual',
 'null_degrees_of_freedom',
 'null_deviance',
 'offset_column',
 'param_names',
 'params',
 'partial_plot',
 'pprint_coef',
 'pr_auc',
 'pred_noise_bandwidth',
 'predict',
 'predict_contributions',
 'predict_leaf_node_assignment',
 'quantile_alpha',
 'r2',
 'r2_stopping',
 'residual_degrees_of_freedom',
 'residual_deviance',
 'respmul',
 'response_column',
 'respsub',
 'rmse',
 'rmsle',
 'rotation',
 'run_time',
 'sample_rate',
 'sample_rate_per_class',
 'save_model_details',
 'save_mojo',
 'score_each_iteration',
 'score_history',
 'score_tree_interval',
 'scoring_history',
 'seed',
 'set_params',
 'show',
 'staged_predict_proba',
 'start',
 'start_time',
 'std_coef_plot',
 'stopping_metric',
 'stopping_rounds',
 'stopping_tolerance',
 'summary',
 'train',
 'training_frame',
 'training_model_metrics',
 'tweedie_power',
 'type',
 'validation_frame',
 'varimp',
 'varimp_plot',
 'weights',
 'weights_column',
 'xval_keys',
 'xvals']
```

The time() give the UTC amount of seconds that have elapsed in floating point values.

In [11]:

```
import time

start = time.time()

for key, num in enumerate(bin_num):

    flights2_gbm = H2OGradientBoostingEstimator(nbins_cats=num, seed=1234)

    flights2_gbm.train(x=predictors, y=response, training_frame=train, validation_fram
e=valid)

end = time.time()

predictionTime = (end-start)
```

```
gbm Model Build progress: |███████████████████████████████████████████████| 100%
gbm Model Build progress: |███████████████████████████████████████████████| 100%
gbm Model Build progress: |███████████████████████████████████████████████| 100%
gbm Model Build progress: |███████████████████████████████████████████████| 100%
gbm Model Build progress: |███████████████████████████████████████████████| 100%
gbm Model Build progress: |███████████████████████████████████████████████| 100%
gbm Model Build progress: |███████████████████████████████████████████████| 100%
gbm Model Build progress: |███████████████████████████████████████████████| 100%
gbm Model Build progress: |███████████████████████████████████████████████| 100%
gbm Model Build progress: |███████████████████████████████████████████████| 100%
```

This is an alternate way of reading in the file for python 3.6

In [12]:

```
flights2_gbm
```

```
Model Details
=============
H2OGradientBoostingEstimator :  Gradient Boosting Machine
Model Key:  GBM_model_python_1580859444109_10


Model Summary:
```

	number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean_depth	min_leaves	m
0	50.0	50.0	110678.0	5.0	5.0	5.0	25.0	3

```
ModelMetricsRegression: gbm
** Reported on train data. **

MSE: 1329.2825288940203
RMSE: 36.459327049385045
MAE: 18.118516320527252
RMSLE: NaN
Mean Residual Deviance: 1329.2825288940203
```

```
ModelMetricsRegression: gbm
** Reported on validation data. **

MSE: 1353.0947850896957
RMSE: 36.784436723833295
MAE: 18.204540718600583
RMSLE: NaN
Mean Residual Deviance: 1353.0947850896957

Scoring History:
```

	timestamp	duration	number_of_trees	training_rmse	training_mae	training_deviance	validation_rmse	validation_m
0	2020-02-04 16:22:55	0.016 sec	0.0	37.031308	18.764566	1371.317777	37.278862	18.815101
1	2020-02-04 16:23:10	15.642 sec	3.0	36.896040	18.616960	1361.317750	37.148286	18.670488
2	2020-02-04 16:23:30	35.627 sec	7.0	36.782714	18.487281	1352.968056	37.043504	18.545082
3	2020-02-04 16:23:55	1 min 0.051 sec	12.0	36.701522	18.389381	1347.001735	36.970735	18.451143
4	2020-02-04 16:24:20	1 min 25.398 sec	17.0	36.646788	18.324858	1342.987095	36.923406	18.389741
5	2020-02-04 16:24:47	1 min 52.681 sec	22.0	36.605118	18.275130	1339.934657	36.889600	18.343364
6	2020-02-04 16:25:13	2 min 18.714 sec	27.0	36.573955	18.239270	1337.654157	36.865003	18.310246
7	2020-02-04 16:25:48	2 min 53.146 sec	34.0	36.529584	18.190719	1334.410508	36.831864	18.267077
8	2020-02-04 16:26:19	3 min 24.133 sec	40.0	36.497495	18.156603	1332.067112	36.808323	18.236518
9	2020-02-04 16:26:49	3 min 54.804 sec	46.0	36.473456	18.131413	1330.312994	36.792856	18.214934
10	2020-02-04 16:27:09	4 min 14.821 sec	50.0	36.459327	18.118516	1329.282529	36.784437	18.204541

```
Variable Importances:
```

	variable	relative_importance	scaled_importance	percentage
0	ORIGIN_AIRPORT	319118368.0	1.000000	0.320754
1	MONTH	198638896.0	0.622461	0.199657
2	DESTINATION_AIRPORT	194843408.0	0.610568	0.195842
3	AIRLINE	132232376.0	0.414368	0.132910
4	FLIGHT_NUMBER	76783288.0	0.240611	0.077177
5	DAY_OF_WEEK	67829360.0	0.212552	0.068177
6	DISTANCE	5454050.5	0.017091	0.005482

Out[12]:

In [26]:

```
predictionTime
```

Out[26]:

```
2393.457242488861
```

In [61]:

```
print('The minutes to run the above code on 5.8 million observations using H2O GBM wit
h 10 bins: ', predictionTime/60)
```

```
The minutes to run the above code on 5.8 million observations using H2O GBM with 10 bi
ns:  39.89095404148102
```

In [53]:

```
import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn import preprocessing, tree

from sklearn.metrics import classification_report, confusion_matrix
```

In [81]:

```
flights = pd.read_csv('flights.csv', encoding='unicode_escape')
```

In [82]:

```
type(flights)
```

Out[82]:

```
pandas.core.frame.DataFrame
```

In [83]:

```
flights.dtypes
```

Out[83]:

```
YEAR                     int64
MONTH                    int64
DAY                      int64
DAY_OF_WEEK              int64
AIRLINE                 object
FLIGHT_NUMBER            int64
TAIL_NUMBER             object
ORIGIN_AIRPORT          object
DESTINATION_AIRPORT     object
SCHEDULED_DEPARTURE      int64
DEPARTURE_TIME         float64
```

```
DEPARTURE_DELAY        float64
TAXI_OUT               float64
WHEELS_OFF             float64
SCHEDULED_TIME         float64
ELAPSED_TIME           float64
AIR_TIME               float64
DISTANCE                 int64
WHEELS_ON              float64
TAXI_IN                float64
SCHEDULED_ARRIVAL        int64
ARRIVAL_TIME           float64
ARRIVAL_DELAY          float64
DIVERTED                 int64
CANCELLED                int64
CANCELLATION_REASON     object
AIR_SYSTEM_DELAY       float64
SECURITY_DELAY         float64
AIRLINE_DELAY          float64
LATE_AIRCRAFT_DELAY    float64
WEATHER_DELAY          float64
dtype: object
```

In [85]:

```
flights = flights.astype({"YEAR":'category', "MONTH":'category',"DAY_OF_WEEK":'categor
y',"FLIGHT_NUMBER":'category',"DISTANCE":'category'})
```

In [86]:

```
flights.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5819079 entries, 0 to 5819078
Data columns (total 31 columns):
YEAR                   category
MONTH                  category
DAY                    int64
DAY_OF_WEEK            category
AIRLINE                object
FLIGHT_NUMBER          category
TAIL_NUMBER            object
ORIGIN_AIRPORT         object
DESTINATION_AIRPORT    object
SCHEDULED_DEPARTURE    int64
DEPARTURE_TIME         float64
DEPARTURE_DELAY        float64
TAXI_OUT               float64
WHEELS_OFF             float64
SCHEDULED_TIME         float64
ELAPSED_TIME           float64
AIR_TIME               float64
DISTANCE               category
WHEELS_ON              float64
TAXI_IN                float64
SCHEDULED_ARRIVAL      int64
ARRIVAL_TIME           float64
ARRIVAL_DELAY          float64
DIVERTED               int64
CANCELLED              int64
CANCELLATION_REASON    object
AIR_SYSTEM_DELAY       float64
```

```
SECURITY_DELAY          float64
AIRLINE_DELAY           float64
LATE_AIRCRAFT_DELAY     float64
WEATHER_DELAY           float64
dtypes: category(5), float64(16), int64(5), object(5)
memory usage: 1.2+ GB
```

In [87]:

```
X = flights[['YEAR','MONTH','DAY_OF_WEEK','FLIGHT_NUMBER','DISTANCE']]

y = flights['DEPARTURE_DELAY']
```

```
(4655263, 5)
(1163816, 5)
(4655263,)
(1163816,)
```

In [95]:

```
flightsXY = pd.concat([X,y], axis=1)
```

In [97]:

```
flightsXY.shape
```

Out[97]:

```
(5819079, 6)
```

In [99]:

```
flightsXY = flightsXY.dropna()

flightsXY.shape
```

Out[99]:

```
(5732926, 6)
```

In [102]:

```
flightsXY
```

Out[102]:

	YEAR	MONTH	DAY_OF_WEEK	FLIGHT_NUMBER	DISTANCE	DEPARTURE_DELAY
0	2015	1	4	98	1448	-11.0
1	2015	1	4	2336	2330	-8.0
2	2015	1	4	840	2296	-2.0
3	2015	1	4	258	2342	-5.0
4	2015	1	4	135	1448	-1.0
...	...	...	...	...	...	...
5819074	2015	12	4	688	2611	-4.0
5819075	2015	12	4	745	1617	-4.0
5819076	2015	12	4	1503	1598	-9.0
5819077	2015	12	4	333	1189	-6.0
5819078	2015	12	4	839	1576	15.0

5732926 rows × 6 columns

In [103]:

```
X = flightsXY.iloc[:,0:5].values

y = flightsXY.iloc[:,5].values


X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=2
0)


print(X_train.shape)

print(X_test.shape)

print(y_train.shape)

print(y_test.shape)
```

```
(4586340, 5)
(1146586, 5)
(4586340,)
(1146586,)
```

In [104]:

```
X_train
```

Out[104]:

```
array([[2015, 7, 3, 4098, 946],
       [2015, 5, 3, 2981, 135],
       [2015, 1, 6, 4096, 472],
       ...,
       [2015, 12, 3, 246, 991],
       [2015, 7, 5, 5046, 83],
       [2015, 3, 3, 3380, 351]], dtype=object)
```

In [105]:

```
y_train
```

Out[105]:

```
array([363.,   3.,  -6., ...,  -4.,  -3.,  20.])
```

In [106]:

```
knn = KNeighborsClassifier()

knn.fit(X_train,

        y_train)
```

Out[106]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [107]:

```
y_pred = knn.predict(X_test)
```

In [109]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy: ',accuracy_score(y_test, y_pred))
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
Accuracy:  0.06603516875315066
```

In [ ]: