02Statistical_modeling

December 19, 2019

1 02 Statistical modeling

Author: Miao Cai miao.cai@slu.edu

1.1 Statistical modeling

We then use four different models to model the risk during the trip:

- Logistic regression
- Poisson regression
- XGBoost
- Deep learning (Neural networks)

1.2 import packages and read data

[1]:
```
# !pip install h2o
import numpy as np
import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

print("numpy version:", np.__version__)
print("h2o version:", h2o.__version__)
```

```
Checking whether there is an H2O instance running at http://localhost:54321...
not found.
Attempting to start a local H2O server...
  Java Version: openjdk version "11.0.2" 2019-01-15; OpenJDK Runtime Environment
18.9 (build 11.0.2+9); OpenJDK 64-Bit Server VM 18.9 (build 11.0.2+9, mixed
mode)
  Starting server from /Users/miaocai/anaconda3/envs/py36/lib/python3.6/site-
packages/h2o/backend/bin/h2o.jar
  Ice root: /var/folders/ng/t3l4gg0s0l3987687051v8vm0000gn/T/tmpunnpzfy6
  JVM stdout: /var/folders/ng/t3l4gg0s0l3987687051v8vm0000gn/T/tmpunnpzfy6/h2o_m
iaocai_started_from_python.out
  JVM stderr: /var/folders/ng/t3l4gg0s0l3987687051v8vm0000gn/T/tmpunnpzfy6/h2o_m
```

```
iaocai_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.
Warning: Your H2O cluster version is too old (10 months and 24 days)! Please
download and install the latest version from http://h2o.ai/download/

--------------------------  ----------------------------------------
H2O cluster uptime:         02 secs
H2O cluster timezone:       America/Chicago
H2O data parsing timezone:  UTC
H2O cluster version:        3.22.1.3
H2O cluster version age:    10 months and 24 days !!!
H2O cluster name:           H2O_from_python_miaocai_fiaorf
H2O cluster total nodes:    1
H2O cluster free memory:    2 Gb
H2O cluster total cores:    8
H2O cluster allowed cores:  8
H2O cluster status:         accepting new members, healthy
H2O connection url:         http://127.0.0.1:54321
H2O connection proxy:
H2O internal security:      False
H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
Python version:             3.6.7 final
--------------------------  ----------------------------------------


numpy version: 1.15.4
h2o version: 3.22.1.3
```

[2]:
```
df = h2o.import_file('https://raw.githubusercontent.com/caimiao0714/
 ↪optimization_stats_case_study/master/data/simulated_data.csv')
df[df['y']  > 0,'y_binary'] = 1
df[df['y'] == 0,'y_binary'] = 0
df['y_binary'] = df['y_binary'].asfactor()
df.head(5)
```

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

[2]:

[3]:
```
lk = h2o.import_file('https://raw.githubusercontent.com/caimiao0714/
 ↪optimization_stats_case_study/master/data/links_traffic_precipitation.csv')
lk.head(5)
```

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

[3]:

1.2.1 Split into train and test sets

[4]:
```
df_splits = df.split_frame(ratios = [0.7, 0.15], seed = 123)

df_train = df_splits[0]
df_test  = df_splits[1]
df_valid = df_splits[2]

print(str(df_train.nrow) + " rows in training set;\n" +
      str(df_test.nrow) + " rows in test set;\n" +
      str(df_valid.nrow) + " rows in validation set.")
```

```
7021 rows in training set;
1482 rows in test set;
1497 rows in validation set.
```

1.3 Logistic regression

[5]:
```
fit_logit = H2OGeneralizedLinearEstimator(family='binomial',
                                          model_id='fit_logit')
fit_logit.train(x = ['Precipitation', 'Traffic', 'Distance'],
                y = 'y_binary',
                training_frame = df_train)
logit_test_fit = fit_logit.model_performance(df_test)
fit_logit._model_json['output']['coefficients_table']
```

```
glm Model Build progress: || 100%
Coefficients: glm coefficients


names          coefficients    standardized_coefficients
-------------  --------------  ---------------------------
Intercept      -3.60444        -2.14407
Distance       0.00100767      0.0322657
Precipitation  0.25638         0.0920454
Traffic        0.830543        0.187609
```

[5]:

[6]:
```
print("Logistic regression model evaluation:")
print("train AUC: " + str(fit_logit.auc()))
print("test  AUC: " + str(logit_test_fit.auc()))
print("---")
print("train Accuracy" + str(fit_logit.accuracy()))
print("test  Accuracy" + str(logit_test_fit.accuracy()))
print("---")
print("train MSE" + str(fit_logit.mse()))
```

```
print("test  MSE" + str(logit_test_fit.mse()))
print("---")
print("train R-square: " + str(fit_logit.r2()))
print("test  R-square: " + str(logit_test_fit.r2()))
```

```
Logistic regression model evaluation:
train AUC: 0.5596341292919064
test  AUC: 0.5638801871833545
---
train Accuracy[[0.18502530292639058, 0.8936048995869534]]
test  Accuracy[[0.17768208465318305, 0.8940620782726046]]
---
train MSE0.09478002969662627
test  MSE0.09376565320196198
---
train R-square: 0.004278462347631851
test  R-square: 0.004429388061521045
```

1.4 Poisson regression

[7]:
```
fit_poisson = H2OGeneralizedLinearEstimator(family='Poisson',
                                            model_id='fit_poisson')
fit_poisson.train(x = ['Precipitation', 'Traffic', 'Distance'],
                  #offset_column = 'Distance',
                  y = 'y',
                  training_frame = df_train)
poisson_test_fit = fit_poisson.model_performance(df_test)
fit_poisson._model_json['output']['coefficients_table']
```

```
glm Model Build progress: || 100%
Coefficients: glm coefficients


names          coefficients    standardized_coefficients
-------------  --------------  ---------------------------
Intercept      -4.37185        -2.10205
Distance       0.00174692      0.0559369
Precipitation  0.334264        0.120008
Traffic        0.947354        0.213995
```

[7]:

[8]:
```
print("Poisson regression model evaluation:")
print("train MSE: " + str(fit_poisson.mse()))
print("test  MSE: " + str(poisson_test_fit.mse()))
print("---")
print("train R-square: " + str(fit_poisson.r2()))
```

```
print("test  R-square: " + str(poisson_test_fit.r2()))
```

```
Poisson regression model evaluation:
train MSE: 0.16471763615686122
test  MSE: 0.17174430698927012
---
train R-square: 0.006623137684569902
test  R-square: 0.0043653505149615635
```

1.5 XGBoost

[9]:
```
from h2o.estimators import H2OXGBoostEstimator
xgboost_params = {
      "ntrees" : 50,
      "max_depth" : 5,
      "learn_rate" : 0.001,
      "sample_rate" : 0.7,
      "col_sample_rate_per_tree" : 0.9,
      "min_rows" : 5,
      "seed": 4241,
      "score_tree_interval": 10
}
fit_xgboost = H2OXGBoostEstimator(**xgboost_params)
fit_xgboost.train(x = ['Precipitation', 'Traffic', 'Distance'],
                  y = 'y_binary',
                  training_frame = df_train,
                  validation_frame = df_valid)
```

```
xgboost Model Build progress: || 100%
```

[10]:
```
xgboost_test_fit = fit_xgboost.model_performance(df_test)
print("XGBoost regression model evaluation:")
print("train AUC: " + str(fit_xgboost.auc()))
print("test  AUC: " + str(xgboost_test_fit.auc()))
print("---")
print("train Accuracy" + str(fit_xgboost.accuracy()))
print("test  Accuracy" + str(xgboost_test_fit.accuracy()))
print("---")
print("train MSE" + str(fit_xgboost.mse()))
print("test  MSE" + str(xgboost_test_fit.mse()))
print("---")
print("train R-square: " + str(fit_xgboost.r2()))
print("test  R-square: " + str(xgboost_test_fit.r2()))
```

```
XGBoost regression model evaluation:
train AUC: 0.6024401326114551
```

```
test  AUC: 0.5456138569826353
---
train Accuracy[[0.4882012605667114, 0.8933200398803589]]
test  Accuracy[[0.48741114139556885, 0.8940620782726046]]
---
train MSE0.2352502407526602
test  MSE0.23521980545603724
---
train R-square: -1.4714460652217607
test  R-square: -1.4974808755773372
```

1.6 Neural networks

[11]:
```
from h2o.estimators.deeplearning import H2OAutoEncoderEstimator,␣
↪H2ODeepLearningEstimator
```

[12]:
```
fit_DL = H2ODeepLearningEstimator(epochs = 1000,
                                  # hidden = [10, 10],
                                  model_id = 'Deep learning',
                                  seed = 1)
fit_DL.train(x = ['Precipitation', 'Traffic', 'Distance'],
             y = 'y_binary',
             training_frame = df_train,
             validation_frame = df_valid)
```

```
deeplearning Model Build progress: || 100%
```

[13]:
```
DL_test_fit = fit_DL.model_performance(df_test)
print("Deep learning model evaluation:")
print("train AUC: " + str(fit_DL.auc()))
print("test  AUC: " + str(DL_test_fit.auc()))
print("---")
print("train Accuracy" + str(fit_DL.accuracy()))
print("test  Accuracy" + str(DL_test_fit.accuracy()))
print("---")
print("train MSE" + str(fit_DL.mse()))
print("test  MSE" + str(DL_test_fit.mse()))
print("---")
print("train R-square: " + str(fit_DL.r2()))
print("test  R-square: " + str(DL_test_fit.r2()))
```

```
Deep learning model evaluation:
train AUC: 0.5743293556716631
test  AUC: 0.5327087442472058
---
train Accuracy[[0.5431840674778821, 0.8936048995869534]]
test  Accuracy[[0.3826870843035134, 0.8940620782726046]]
---
```

```
train MSE0.09486584281605369
test  MSE0.09535136906060053
---
train R-square: 0.003376943625802875
test  R-square: -0.012407183261082366
```

1.7 Prediction for links data

[14]:
```
risk_logit = fit_logit.predict(lk).as_data_frame(True).p1.tolist()
risk_poisson = fit_poisson.predict(lk).as_data_frame(True).predict.tolist()
risk_xgboost = fit_xgboost.predict(lk).as_data_frame(True).p1.tolist()
risk_DL = fit_DL.predict(lk).as_data_frame(True).p1.tolist()

lk_risks = lk.cbind(h2o.H2OFrame(risk_logit).set_names(['risk_logit'])).\
             cbind(h2o.H2OFrame(risk_poisson).set_names(['risk_poisson'])).\
             cbind(h2o.H2OFrame(risk_xgboost).set_names(['risk_xgboost'])).\
             cbind(h2o.H2OFrame(risk_DL).set_names(['risk_DL']))
lk_risks.head(5)
```

```
glm prediction progress: || 100%
glm prediction progress: || 100%
xgboost prediction progress: || 100%
deeplearning prediction progress: || 100%
Parse progress: || 100%
Parse progress: || 100%
Parse progress: || 100%
Parse progress: || 100%
```

[14]:

[15]:
```
lk_risks.as_data_frame().to_csv('lk_risks.csv')
```