# Приложение 1. Deep Water

## 1. Установка

На момент написания данного текста библиотека Deep Water не была официально представлена, поэтому есть всего три варианта ее установки: сборка из исходных кодов, использование H2O Deep Water Amazon Machine Image (AMI) и запуск образа Docker.

### 1.1. Сборка из исходных кодов

Инструкцию можно найти по ссылке https://github.com/h2oai/deepwater. Доступны варианты сборки под разные платформы; можно использовать различные библиотеки линейной алгебры, включая MKL, OpenBLAS и ATLAS, а также CUDA.

### 1.2. Amazon Machine Image

Для удобства компания H2O.ai создала Deep Water AMI, которые можно устанавливать на GPU-сервера Amazon. Эти образы постоянно обновляются. Актуальную информацию можно найти по ссылке http://bit.ly/2viVKmw. Дополнительные сведения о GPU-серверах Amazon доступны на странице http://amzn.to/1OpWzIO.

### 1.3. Образ Docker

Образ Docker с поддержкой GPU размещен на Docker Hub. Для его использования нужен ПК под управлением ОС семейства Linux с хотя бы одной видеокартой и установленными приложениями Docker и nvidia-docker. Дополнительную информацию можно найти на странице https://github.com/h2oai/deepwater/blob/master/README.md.

### 1.4. Примеры данных

Примеры, рассмотренные в этом руководстве, подразумевают наличие наборов данных в папке *bigdata*, которая расположена в той же папке, откуда запущена библиотека H2O. После клонирования репозитория https://github.com/h2oai/deepwater на свой ПК запустите команду ./gradlew syncBigdataLaptop (Linux) или gradlew syncBigdataLaptop (Windows), чтобы запустить скачивание данных.

## 2. Обзор библиотеки Deep Water

Deep Water представляет собой дополнение для библиотеки H2O, позволяющее использовать алгоритмы глубокого обучения помимо имеющейся в самой библиотеке масштабируемой распределенной реализации многослойного перцептрона (которая работает с данными, помещающимися в ОЗУ).

### 2.1. Глубокое обучение в библиотеке H2O

В библиотеке H2O реализован быстрый и точный алгоритм глубокого обучения, который широко используется в мире. Как и остальные методы машинного обучения в H2O, он обладает высокой скоростью работы и доступен при помощи интерфейсов для языков R, Python и Java, а также через веб-интерфейс Flow. Ключевые особенности:

* настройки процесса обучения модели: можно выбирать вид распределения (распределение Бернулли, мультиномиальное, пуассоновское, гамма-распределение, распределение Твиди, Лапласа, Хьюбера, квантильное и нормальное распределение), функцию потерь (перекрестная энтропия, квадратичная и абсолютная ошибка, функция потерь Хьюбера), скорость обучения, уменьшение скорости обучения, импульс, размер мини-выборки, способ инициализации;
* автоматическая и гибкая обработка данных: стандартизация, бинарное кодирование, указание весов наблюдений, балансировка классов, исключение переменных с нулевой дисперсией, обработка разреженных данных;
* настройки для борьбы с переобучением: перекрестная проверка, регуляризация, дропаут, ранняя остановка, контрольные точки, подбор значений гиперпараметров;
* глубокие автокодировщики для обучения на неразмеченных данных: создание признаков (использование выходных значений нейронов скрытых слоев), определение аномалий.

Полный перечень возможностей можно найти в официальной документации и в буклете http://bit.ly/2rnigXd.

### 2.2. Современные тенденции в глубоком обучении

Глубокое обучение как практическая дисциплина и как область знаний значительно изменилось с момента начала разработки Deep Water. Сверточные и рекуррентные нейронные сети (в том числе блоки, используемые в архитектуре Inception, и остаточные (residual) нейросети) ставят рекорды во многих областях ИИ, включая компьютерное зрение и обработку звуков, речи и естественного языка. Использование GPU для обучения таких сложных нейронных сетей выглядит многообещающим, и производительность GPU продолжает расти. Появилось и развивается множество фреймворков для глубокого обучения с поддержкой GPU, в том числе TensorFlow, MXNet, Caffe, Theano и Torch.

### 2.3. Почему нужно использовать Deep Water

Deep Water является дополнением к библиотеке H2O и предоставляет следующие возможности:

* интеграция с производительными и масштабируемыми фремворками для глубокого обучения (TensorFlow, MXNet, Caffe), поддержка использования GPU, современные нейросетевые архитектуры (такие как VGG, ResNet), возможность обучения собственных или предварительно обученных сетей;
* платформа для машинного обучения: модели Deep Water можно сравнивать с другими моделями, созданными в H2O, а также объединять их в ансамбли;
* простота использования, наличие API для R, Python и Java, доступ через веб-интерфейс Flow;
* развертывание: модели Deep Water можно использовать точно так же, как и другие модели, созданные при помощи H2O. В частности, они могут быть экспортированы в формате MOJO, который используется в основанных на JVM языках. Также есть возможность добавления «оберток» для других языков. Дополнительную информацию можно найти по ссылке http://bit.ly/2hprGjY.

## 3. Начало работы: набор данных MNIST

Следующий пример описывает начало работы с Deep Water. В нем показано работа с API и применение возможностей основной библиотеки H2O. Используется набор данных MNIST; обучается многослойная нейронная сеть с дропаутом для слоя входных данных, перекрестной проверкой, ранней остановкой и с использованием GPU (по умолчанию).

Пример на языке Python:

import h2o

from h2o.estimators.deepwater import H2ODeepWaterEstimator

# Запуск или соединение с H2O

h2o.init()

# Импортирование и преобразование данных

train = h2o.import\_file("bigdata/laptop/mnist/train.csv.gz")

# Набор признаков для включения в модель

features = list(range(0, 784))

target = 784

train[target] = train[target].asfactor()

# Создание модели

model = H2ODeepWaterEstimator(

epochs = 100,

activation = "Rectifier",

hidden = [200, 200],

ignore\_const\_cols = False,

mini\_batch\_size = 256,

input\_dropout\_ratio = 0.1,

hidden\_dropout\_ratios = [0.5, 0.5],

stopping\_rounds = 3,

stopping\_tolerance = 0.05,

stopping\_metric = "misclassification",

score\_interval = 2,

score\_duty\_cycle = 0.5,

score\_training\_samples = 1000,

score\_validation\_samples = 1000,

nfolds = 5,

gpu = True,

seed = 1234)

model.train(x = features, y = target, training\_frame = train)

# Оценка качества модели

model.show()

print(model.scoring\_history())

### 3.1. Бекенды

По умолчанию в качестве бекенда используется mxnet. Другой вариант можно выбрать при помощи параметра backend.

Пример на языке Python:

model = H2ODeepWaterEstimator(

epochs = 100,

activation = "Rectifier",

hidden = [200, 200],

ignore\_const\_cols = False,

mini\_batch\_size = 256,

input\_dropout\_ratio = 0.1,

hidden\_dropout\_ratios = [0.5, 0.5],

stopping\_rounds = 3,

stopping\_tolerance = 0.05,

stopping\_metric = "misclassification",

score\_interval = 2,

score\_duty\_cycle = 0.5,

score\_training\_samples = 1000,

score\_validation\_samples = 1000,

nfolds = 5,

gpu = True,

seed = 1234,

backend = "tensorflow")

### 3.2. CPU и GPU

GPU используются для вычислений по умолчанию, но это не является необходимым. Задав параметр gpu = False, можно запустить вычисления только с использование CPU.

Пример на языке Python:

model = H2ODeepWaterEstimator(

epochs = 100,

activation = "Rectifier",

hidden = [200, 200],

ignore\_const\_cols = False,

mini\_batch\_size = 256,

input\_dropout\_ratio = 0.1,

hidden\_dropout\_ratios = [0.5, 0.5],

stopping\_rounds = 3,

stopping\_tolerance = 0.05,

stopping\_metric = "misclassification",

score\_interval = 2,

score\_duty\_cycle = 0.5,

score\_training\_samples = 1000,

score\_validation\_samples = 1000,

nfolds = 5,

gpu = False,

seed = 1234)

### Использование Deep Water с R

В этом руководстве приведены примеры с использования языка Python, но также доступно API для R.

Пример на языке R:

library(h2o)

# Запуск или соединение с H2O

h2o.init()

# Импортирование и преобразование данных

train <- h2o.importFile("bigdata/laptop/mnist/train.csv.gz")

target <- "C785"

features <- setdiff(names(train), target)

train[target] <- as.factor(train[target])

# Создание модели

model <- h2o.deepwater(

x = features,

y = target,

training\_frame = train,

epochs = 100,

activation = "Rectifier",

hidden = c(200, 200),

ignore\_const\_cols = FALSE,

mini\_batch\_size = 256,

input\_dropout\_ratio = 0.1,

hidden\_dropout\_ratios = c(0.5, 0.5),

stopping\_rounds = 3,

stopping\_tolerance = 0.05,

stopping\_metric = "misclassification",

score\_interval = 2,

score\_duty\_cycle = 0.5,

score\_training\_samples = 1000,

score\_validation\_samples = 1000,

nfolds = 5,

gpu = TRUE,

seed = 1234)

# Оценка качества модели

summary(model)

Полные версии следующих примеров на языке Python можно найти в Jupyter-ноутбуках по ссылке http://bit.ly/2f9KCSS.

## 4. Классификация изображений

### 4.1. Данные

Библиотека Deep Water позволяет работать со стандартными таблицами H2O в следующих форматах:

* набор данных – обычная таблица с числовыми и категориальными признаками;
* изображения – таблица с двумя столбцами, в которой первый столбец содержит ссылки на изображения, а второй – значения целевой переменной (метки классов для задач классификации) для обучения на размеченных данных.

Формат задается при помощи параметра problem\_type с возможными значениями "dataset", "image" или "auto" (вариант по умолчанию, при котором решение принимается автоматически).

### 4.2. Параметры изображений

Для корректной обработки изображений могут быть указаны следующие основные параметры:

* image\_shape – список целых чисел, которые задают ширину и высоту изображения;
* channels – целое число, соответствующее количеству каналов (3 для изображений в RGB-формате);
* mean\_image\_file – строка, содержащая пусть к усредненному изображению, которое используется для нормализации данных.

Пример на языке Python:

import h2o

from h2o.estimators.deepwater import H2ODeepWaterEstimator

# Запуск или соединение с H2O

h2o.init()

# Импортирование и преобразование данных

train = h2o.import\_file("bigdata/laptop/deepwater/imagenet/cat\_dog\_mouse.csv")

# Создание модели

model = H2ODeepWaterEstimator(

epochs = 10,

network = "lenet",

problem\_type = "image",

image\_shape = [28, 28],

channels = 3)

model.train(x = [0], y = 1, training\_frame = train)

# Оценка качества модели

model.show()

### 4.3. Предварительно созданные архитектуры

В готовом виде доступны следующие известные архитектуры для классификации изображений:

* LeNet;
* AlexNet;
* VGG;
* GoogLeNet;
* Inception-bn;
* ResNet.

Список будет пополняться.

### 4.4. Архитектуры, создаваемые пользователем

Если задать параметр network = "user", пользователь может сам определять архитектуру используемой нейросети. Создание архитектуры (графа вычислений) происходит с использованием API библиотеки, которая выступает в роли бекенда. Нейросеть сохраняется и затем может быть загружена при помощи параметра network\_definition\_file. Процесс аналогичен использованию предварительно обученной сети, но при этом не указываются параметры нейросети (веса и смещения). Примеры приведены в разделе 4.5.

Mxnet: используется API mxnet.symbol и класс Symbol (http://bit.ly/2hqnkZT). За сохранение нейросети отвечает метод Symbol.save.

Tensorflow: используется класс tf.Graph или высокоуровневый API, например, библиотека Keras (https://keras.io/). Сохранение происходит при помощи класса tf.train.Saver (см. http://bit.ly/2ldnpgE и http://bit.ly/2fatrki) и метода tf.train.export\_meta\_graph().

### 4.5. Предварительно обученные нейросети

В Deep Water могут быть загружены параметры нейросети (веса и смещения) вместе с описанием архитектуры, как показано ниже.

Пример на языке Python (mxnet):

model = H2ODeepWaterEstimator(

epochs = 100,

image\_shape =[28, 28],

backend = "mxnet",

network = "user",

network\_definition\_file = "/path/to/lenet.json",

network\_parameters\_file = "/path/to/lenet-100epochsparams.txt")

Пример на языке Python (Tensorflow):

model = H2ODeepWaterEstimator(

epochs = 100,

image\_shape = [28, 28],

backend = "tensorflow",

network = "user",

network\_definition\_file = "/path/to/lenet\_28x28x3\_3.meta",

network\_parameters\_file = "/path/to/lenet-100epochs")

## 5. Веб-интерфейс Flow

Доступ к функциям Deep Water можно получить также посредством веб-интерфейса Flow, как показано на следующих иллюстрациях.

@Рис. 5.1. Импортирование данных

@Рис. 5.2. Парсинг данных

@Рис. 5.3. Просмотр данных

@Рис. 5.4. Создание модели

@Рис. 5.5. Выбор бекенда

@Рис. 5.6. Использование GPU

## 6. Поиск по сетке

Deep Water позволяет использовать возможности подбора значений гиперпараметров путем поиска по сетке, реализованные в библиотеке H2O. Можно указывать тип поиска и настраивать раннюю остановку. Все настройки передаются параметру search\_criteria в виде словаря:

* strategy: "Cartesian" (по умолчанию) или "RandomDiscrete";
* stopping\_metric – метрика, используемая для ранней остановки;
* stopping\_rounds – обучение останавливается, если используемая метрика не улучшается после заданного числа итераций оценивания (значение 0 отключает эту опцию);
* stopping\_tolerance – задает требуемую величину улучшения по метрике качества; остановка происходит при отсутствии улучшения хотя бы на эту величину.

Более подробно подбор значений гиперпараметров рассматривается в сообщении http://bit.ly/2u6NPJN.

### 6.1. Полный перебор

В данном примере проверяется два варианта конфигурации скрытых слоев и две скорости обучения, то есть всего будет построено четыре модели.

Пример на языке Python:

# Импортирование и преобразование данных

train = h2o.import\_file("bigdata/laptop/mnist/train.csv.gz")

features = list(range(0, 784))

target = 784

train[target] = train[target].asfactor()

# Набор проверяемых значений гиперпараметров

hidden\_opt = [[200, 200], [1024, 1024]]

learn\_rate\_opt = [1e-6, 1e-5]

hyper\_parameters = {"hidden": hidden\_opt, "learning\_rate": learn\_rate\_opt}

# Создание модели и поиск по сетке

from h2o.grid.grid\_search import H2OGridSearch

model\_grid = H2OGridSearch(

H2ODeepWaterEstimator,

hyper\_params = hyper\_parameters)

model\_grid.train(

x = features,

y = target,

training\_frame = train,

epochs = 100,

activation = "Rectifier",

ignore\_const\_cols = False,

mini\_batch\_size = 256,

input\_dropout\_ratio = 0.1,

hidden\_dropout\_ratios = [0.5, 0.5],

stopping\_rounds = 3,

stopping\_tolerance = 0.05,

stopping\_metric = "misclassification",

score\_interval = 2,

score\_duty\_cycle = 0.5,

score\_training\_samples = 1000,

score\_validation\_samples = 1000,

nfolds = 5,

gpu = True,

seed = 1234)

# Оценка качества модели

print(model\_grid)

### 6.2. Случайный поиск

Пространство возможных значений гиперпараметров может быть слишком большим для полного перебора, поэтому за заданный промежуток времени можно случайным образом проверить лишь некоторые из комбинаций. В следующем примере настраиваемыми гиперпараметрами являются конфигурация скрытых слоев, скорость обучения и дропаут для входного слоя. Время поиска ограничено пятью минутами.

Пример на языке Python:

# Набор проверяемых значений гиперпараметров

hidden\_opt = [[200,200], [1024,1024], [1024, 1024, 2048],

[200, 200, 200], [300, 300]]

learn\_rate\_opt = [1e-6, 1e-5, 1e-3, 5e-3]

in\_drop\_opt = [0.1, 0.2, 0.3]

hyper\_parameters = {"hidden": hidden\_opt,

"learning\_rate":learn\_rate\_opt,

"input\_dropout\_ratio": in\_drop\_opt}

search\_criteria = {"strategy": "RandomDiscrete",

"max\_models": 10,

"max\_runtime\_secs": 300,

"seed": 1234}

# Создание модели и поиск по сетке

from h2o.grid.grid\_search import H2OGridSearch

model\_grid = H2OGridSearch(

H2ODeepWaterEstimator,

hyper\_params = hyper\_parameters,

search\_criteria = search\_criteria)

model\_grid.train(

x = features,

y = target,

training\_frame = train,

epochs = 100,

activation = "Rectifier",

ignore\_const\_cols = False,

mini\_batch\_size = 256,

hidden\_dropout\_ratios = [0.5, 0.5],

stopping\_rounds = 3,

stopping\_tolerance = 0.05,

stopping\_metric = "misclassification",

score\_interval = 2,

score\_duty\_cycle = 0.5,

score\_training\_samples = 1000,

score\_validation\_samples = 1000,

nfolds = 5,

gpu = True,

seed = 1234)

# Оценка качества модели

print(model\_grid)

## 7. Контрольные точки

Контрольные точки полезно использовать для сохранения моделей (т.е. их состояния в процессе обучения), если обучение занимает много времени, а также для того, чтобы в дальнейшем продолжать обучение (иногда с другими параметрами). В примере ниже модель обучается в течение 20 эпох и сохраняется при помощи метода h2o.save\_model. Затем она загружается при помощи метода h2o.load\_model, после чего обучение продолжается.

Пример на языке Python:

# Импортирование и преобразование данных

train = h2o.import\_file("bigdata/laptop/mnist/train.csv.gz")

valid = h2o.import\_file("bigdata/laptop/mnist/test.csv.gz")

features = list(range(0,784))

target = 784

train[target] = train[target].asfactor()

valid[target] = valid[target].asfactor()

# Создание модели

model = H2ODeepWaterEstimator(

epochs = 20,

activation = "Rectifier",

hidden = [200, 200],

ignore\_const\_cols = False,

mini\_batch\_size = 256,

input\_dropout\_ratio = 0.1,

hidden\_dropout\_ratios = [0.5, 0.5],

stopping\_rounds = 3,

stopping\_tolerance = 0.05,

stopping\_metric = "misclassification",

score\_interval = 2,

score\_duty\_cycle = 0.5,

score\_training\_samples = 1000,

score\_validation\_samples = 1000,

gpu = True,

seed = 1234)

model.train(

x = features,

y = target,

training\_frame = train,

validation\_frame = valid)

# Оценка качества модели

model.show()

print(model.scoring\_history())

# Сохранение модели

model\_path = h2o.save\_model(model=model, force=True)

# Загрузка модели

model\_ckpt = h2o.load\_model(model\_path)

# Продолжение обучение с контрольной точки

model\_warm = H2ODeepWaterEstimator(

checkpoint = model\_ckpt.model\_id,

epochs = 100,

activation = "Rectifier",

hidden = [200, 200],

ignore\_const\_cols = False,

mini\_batch\_size = 256,

input\_dropout\_ratio = 0.1,

hidden\_dropout\_ratios = [0.5, 0.5],

stopping\_rounds = 3,

stopping\_tolerance = 0.05,

stopping\_metric = "misclassification",

score\_interval = 2,

score\_duty\_cycle = 0.5,

score\_training\_samples = 1000,

score\_validation\_samples = 1000,

gpu = True,

seed = 1234)

model\_warm.train(

x = features,

y = target,

training\_frame = train,

validation\_frame = valid)

# Оценка качества модели

model\_warm.show()

print(model\_warm.scoring\_history())

## 8. Ансамбли

Модели Deep Water можно объединять в ансамбли с другими моделями, созданными в библиотеке H2O. В данном примере используются три базовые модели: GBM, GLM и Deep Water, объединенные при помощи стекинга. Более подробная информация о стекинге моделей доступна по ссылке http://bit.ly/2ht6M3v.

Пример на языке Python:

import h2o

from h2o.estimators.deepwater import H2ODeepWaterEstimator

from h2o.estimators.gbm import H2OGradientBoostingEstimator

from h2o.estimators.glm import H2OGeneralizedLinearEstimator

from h2o.estimators.stackedensemble import H2OStackedEnsembleEstimator

# Импортирование данных

train = h2o.import\_file("/path/to/train-odd.csv.gz",

destination\_frame = "train.hex")

valid = h2o.import\_file("/path/to/test-odd.csv.gz",

destination\_frame="valid.hex")

features = list(range(0, 784))

target = 784

train[features] = train[features]/255

train[target] = train[target].asfactor()

valid[features] = valid[features]/255

valid[target] = valid[target].asfactor()

nfolds = 5

# GBM

gbm\_model = H2OGradientBoostingEstimator(

distribution = "bernoulli",

ntrees = 100,

nfolds = nfolds,

ignore\_const\_cols = False,

keep\_cross\_validation\_predictions = True,

fold\_assignment = "Modulo")

gbm\_model.train(

x = features,

y = target,

training\_frame = train,

model\_id = "gbm\_model")

gbm\_model.show()

# GLM

glm\_model = H2OGeneralizedLinearEstimator(

family = "binomial",

lambda\_min\_ratio = 0.0001,

alpha = 0.5,

nfolds = nfolds,

ignore\_const\_cols = False,

keep\_cross\_validation\_predictions = True,

fold\_assignment = "Modulo")

glm\_model.train(

x = features,

y = target,

training\_frame = train,

model\_id = "glm\_model")

glm\_model.show()

# Модель Deep Water

dw\_model = H2ODeepWaterEstimator(

epochs = 3,

network = "lenet",

ignore\_const\_cols = False,

image\_shape = [28, 28],

channels = 1,

standardize = False,

seed = 1234,

nfolds = nfolds,

keep\_cross\_validation\_predictions = True,

fold\_assignment = "Modulo")

dw\_model.train(

x = features,

y = target,

training\_frame = train,

model\_id = "dw\_model")

dw\_model.show()

# Ансамбль

stack\_all = H2OStackedEnsembleEstimator(

base\_models = [gbm\_model.model\_id,

glm\_model.model\_id,

dw\_model.model\_id])

stack\_all.train(

x = features,

y = target,

training\_frame = train,

validation\_frame = valid,

model\_id = "stack\_all")

stack\_all.model\_performance()

## 9. Признаки скрытых слоев и меры сходства

Скрытые слои обученной модели могут дать нам полезное представление входных данных в другом признаковом пространстве (альтернативное признаковое описание). У моделей Deep Water есть метод deepfeatures, позволяющий извлекать эти признаки, которые затем можно использовать различными способами. В следующем примере происходит извлечение признаков из скрытого слоя предварительно обученной сверточной нейронной сети. Полученные признаки используются для обучения модели класса GLM.

Пример на языке Python:

# Загрузка нейросети

network\_model = H2ODeepWaterEstimator(

epochs = 0,

mini\_batch\_size = 32,

network="user",

network\_definition\_file = "Inception\_BN-symbol.json",

network\_parameters\_file = "Inception\_BN-0039.params",

mean\_image\_file = "mean\_224.nd",

image\_shape = [224, 224],

channels = 3)

network\_model.train(x = [0], y = 1, training\_frame = train)

# Извлечение признаков

extracted\_features = network\_model.deepfeatures(train, "global\_pool\_output")

print("shape: " + str(extracted\_features.shape))

print(extracted\_features[:5, :3])

# Объединение признаков с целевой переменной и разделение таблицы

extracted\_features["target"] = train[1]

features = [x for x in extracted\_features.columns if x not in ["target"]]

train, valid = extracted\_features.split\_frame(ratios = [0.8])

# Обучение модели класса GLM

glm\_model = H2OGeneralizedLinearEstimator(family= "multinomial")

glm\_model.train(

x = features,

y = "target",

training\_frame = train,

validation\_frame = valid)

# Оценка качества модели

glm\_model.show()

Другое применение таких признаков заключается в их использовании при решении задач обучения на неразмеченных данных, таких как кластеризация. Они выступают в качестве векторного преставления данных, что позволяет вычислять меры сходства (l1- и l2-норму, косинусную меру и квадрат косинусной меры).

Пример на языке Python:

# Разделение на таблицы references и queries

references = extracted\_features[5:, :]

queries = extracted\_features[:3, :]

# Вычисление сходства

similarity = references.distance(queries, "cosine")

# Проверка размерностей

print("references: " + str(references.shape))

print("queries: " + str(queries.shape))

print("similarity: " + str(similarity.shape))

# Вывод результатов

print(similarity.head())

## 10. Поддержка нескольких GPU

Поддержка вычислений на нескольких GPU зависит от используемого бекенда.

## 11. Развертывание моделей

### 11.1. MOJO

Модели могут быть сохранены в бинарном формате MOJO (см. http://bit.ly/2hprGjY), после чего их можно запускать в исполняющей среде Java независимо от H2O. Все, что требуется, это наличие файлов *h2o-genmodel.jar* и *deepwater-all.jar*.

Модель в формате MOJO может быть получена в веб-интерфейсе Flow или при помощи команды вида

model.download\_mojo(path = "/path/to/model\_mojo", get\_genmodel\_jar = True)

### 11.2. Prediction Service Builder

Prediction Service Builder является отдельным веб-приложением, помогающим получать модели в формате MOJO и файлы Web Archive (*war*) для создания сервисов на основе прогностических моделей. Инструкции по сборке Prediction Service Builder можно найти на странице http://bit.ly/2u9OU3B.

Перед тем, как приступить в созданию *war*-файла, убедитесь в наличии у вас файлов *h2o-genmodel.jar* и *deepwater-all.jar*. Их можно получить, выполнив команды:

curl localhost:54321/3/h2o-genmodel.jar > h2o-genmodel.jar

curl localhost:54321/3/deepwater-all.jar > deepwater-all.jar

*war*-файлы можно создавать как с использованием веб-интерфейса Prediction Service Builder, так и из командной строки. Пример:

curl -X POST \

--form mojo=@mojo.zip \

--form jar=@h2o-genmodel.jar \

--form deepwater=@deepwater-all.jar \

localhost:55000/makewar > example.war

Файл example.war может быть запущен при помощи Jetty Runner:

java -jar jetty-runner-9.3.9.M1.jar --port 55001 example.war

Доступ к сервису на основе модели можно получить по адресу http://localhost:55001.