Приложение 2

Ансамбли (стекинг моделей)

Вступление

Ансамблевые методы машинного обучения используют несколько алгоритмов для получения более высокого качества, по сравнению с качеством каждого отдельного алгоритма. Многие из популярных современных методов машинного обучения являются ансамблями моделей (например, «случайный лес» и градиентный бустинг комбинируют предсказания множества «слабых» моделей для получения одной «сильной» модели).

Реализованный в H2O метод вложенных ансамблей (stacked ensembles) является алгоритмом машинного обучения на размеченных данных, который находит оптимальную комбинацию алгоритмов с использованием процедуры под названием стекинг[1] (stacking). В данный момент поддерживается использование стекинга для задач регрессии и бинарной классификации, в дальнейшем планируется добавить поддержку многоклассовой классификации.

Реализация стекинга добавлена в основную библиотеку H2O в версии 3.10.3.1. Реализация в виде отдельного R-пакета h2oEnsemble также остается доступной (https://github.com/h2oai/h2o-3/tree/master/h2o-r/ensemble), но мы рекомендуем использовать версию из основной библиотеки, описанную ниже.

Стекинг / Super Learner

Стекинг, также известный как алгоритм «Super Learning» и вложенная регрессия (stacked regression), представляет собой класс алгоритмов, включающих обучение метамодели второго уровня для нахождения оптимального сочетания базовых моделей. В отличие от бэггинга[2] (bagging) и бустинга (boosting), целью является объединение в ансамбль нескольких сильных и разнообразных по своим свойствам моделей.

---

[1] Более подробно об этом методе можно прочитать по ссылке https://alexanderdyakonov.wordpress.com/2017/03/10/стекинг-stacking-и-блендинг-blending/. – *Прим. перев.*

[2] См., напр.: https://habrahabr.ru/company/ods/blog/324402/. – *Прим. перев.*

Идея стекинга была предложена в 1992 г., но теоретическое обоснование метода отсутствовало до выхода публикации под названием «Super Learner» (http://bit.ly/2eMDyeE) в 2007 г. В ней было показано, что алгоритм «Super Learner» представляет собой асимптотически оптимальную систему машинного обучения.

Существует несколько методов создания ансамблей, которые в целом называют стекингом; особенностью алгоритма «Super Learner» является использование перекрестной проверки для создания так называемых «данных первого уровня» (level-one data), то есть данных, на которых обучается метаалгоритм. Более подробно эта процедура рассмотрена далее.

Алгоритм

Следующие этапы описывают задачи, решаемые при обучении и оценке качества алгоритма «Super Learner». Библиотека H2O автоматизирует большинство из них, и вы можете быстро и легко создавать ансамбли из моделей, предоставляемых данной библиотекой.

1. Конфигурация ансамбля.
   a. Задать список из L базовых алгоритмов (с набором гиперпараметров для каждой модели).
   b. Задать алгоритм метаобучения.
2. Обучение ансамбля.
   a. Обучить каждую из L базовых моделей на обучающей выборке.
   b. Выполнить перекрестную проверку с разбивкой на K блоков для каждой из L базовых моделей и сохранить предсказанные значения для каждого блока, когда он выступает в качестве проверочной выборки.
   c. N значений, предсказанных в ходе выполнения перекрестной проверки для каждой из L базовых моделей, образуют матрицу N × L. Эта матрица вместе с вектором правильных ответов образует «данные первого уровня» (N – число строк в обучающем наборе данных).
   d. Обучить метаалгоритм на данных первого уровня. Ансамблевая модель состоит из L базовых моделей и метамодели, которые затем используются вместе для предсказаний на тестовой выборке.
3. Предсказание на новых данных.
   a. Получить предсказанные значения при помощи базовых алгоритмов (обученных на всей обучающей выборке).
   b. Использовать эти значения в качестве входных данных для метамодели.

Вложенные ансамбли в библиотеке H2O

- *model_id* – пользовательские идентификаторы моделей, по которым к ним можно обращаться.
- *training_frame* – набор данных, используемый для обучения.
- *validation_frame* – набор данных, используемый для оценки качества.
- *base_models* – список идентификаторов моделей, которые используются в ансамбле. Для этих моделей должна использоваться перекрестная проверка с `nfolds` > 1, причем разбивка на блоки обязана быть одинаковой с параметром `keep_cross_validation_folds = TRUE`.

Замечания по поводу base_models:

- одним из способов обеспечения одинаковости блоков является использование fold_assignment = «Modulo» для всех базовых моделей. Также можно получить идентичные блоки, задав fold_assignment = «Random» с одинаковым начальным значением генератора случайных чисел;
- в R параметр base_models может принимать список моделей.

Также в следующих версиях (https://0xdata.atlassian.net/browse/PUBDEV-3743) будет добавлен параметр metalearner, который позволит выбирать тип используемого метаалгоритма. Сейчас в качестве метаалгоритма используется стандартный вариант GLM с неотрицательными весами.

Вы можете следить за ходом разработки по ссылке https://0xdata.atlassian.net/issues/?filter=19301.

Пример

На языке R

```
library(h2o)
h2o.init()

# Импорт обучающей (с бинарной переменной отклика) и тестовой выборки H2O
train <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_train_10k.csv")
test <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_test_5k.csv")

# Предикторы и целевая переменная
y <- «response»
x <- setdiff(names(train), y)

# Для бинарной классификации целевая переменная должна быть фактором
train[, y] <- as.factor(train[, y])
test[, y] <- as.factor(test[, y])

# Количество блоков для перекрестной проверки (используются при создании «данных первого уровня»)
nfolds <- 5

# Есть несколько способов получения списка моделей для стекинга:
# 1. Обучить отдельные модели и собрать их в список
# 2. Обучить несколько моделей, используя h2o.grid
# 3. Использовать h2o.grid несколько раз
# Примечание: блоки должны быть одинаковыми для всех моделей,
# также необходимо сохранить предсказания, полученные в ходе перекрестной проверки


# 1. Создание ансамбля из двух моделей (GBM + RF)

# Обучение и перекрестная проверка GBM
my_gbm <- h2o.gbm(x = x,
                  y = y,
                  training_frame = train,
                  distribution = "bernoulli",
                  ntrees = 10,
                  max_depth = 3,
                  min_rows = 2,
                  learn_rate = 0.2,
```

```
                nfolds = nfolds,
                fold_assignment = "Modulo",
                keep_cross_validation_predictions = TRUE,
                seed = 1)

# Обучение и перекрестная проверка RF
my_rf <- h2o.randomForest(x = x,
                          y = y,
                          training_frame = train,
                          ntrees = 50,
                          nfolds = nfolds,
                          fold_assignment = "Modulo",
                          keep_cross_validation_predictions = TRUE,
                          seed = 1)

# Обучение ансамбля из GBM и RF
ensemble <- h2o.stackedEnsemble(x = x,
                                y = y,
                                training_frame = train,
                                model_id = "my_ensemble_binomial",
                                base_models = list(my_gbm@model_id,
                                                   my_rf@model_id))

# Оценка качества ансамбля на тестовых данных
perf <- h2o.performance(ensemble, newdata = test)

# Сравнение с качеством базовых моделей на тестовых данных
perf_gbm_test <- h2o.performance(my_gbm, newdata = test)
perf_rf_test <- h2o.performance(my_rf, newdata = test)
baselearner_best_auc_test <- max(h2o.auc(perf_gbm_test), h2o.auc(perf_rf_test))
ensemble_auc_test <- h2o.auc(perf)
print(sprintf("Best Base-learner Test AUC:  %s", baselearner_best_auc_test))
print(sprintf("Ensemble Test AUC:  %s", ensemble_auc_test))

# Создание предсказаний для тестовой выборки (если требуется)
pred <- h2o.predict(ensemble, newdata = test)


# 2. Создание и стекинг моделей с использованием h2o.grid

# Гиперпараметры GBM
learn_rate_opt <- c(0.01, 0.03)
max_depth_opt <- c(3, 4, 5, 6, 9)
sample_rate_opt <- c(0.7, 0.8, 0.9, 1.0)
col_sample_rate_opt <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8)
hyper_params <- list(learn_rate = learn_rate_opt,
                     max_depth = max_depth_opt,
                     sample_rate = sample_rate_opt,
                     col_sample_rate = col_sample_rate_opt)

search_criteria <- list(strategy = "RandomDiscrete",
                        max_models = 3,
                        seed = 1)

gbm_grid <- h2o.grid(algorithm = "gbm",
                     grid_id = "gbm_grid_binomial",
                     x = x,
```

```
                    y = y,
                    training_frame = train,
                    ntrees = 10,
                    seed = 1,
                    nfolds = nfolds,
                    fold_assignment = "Modulo",
                    keep_cross_validation_predictions = TRUE,
                    hyper_params = hyper_params,
                    search_criteria = search_criteria)

# Обучение ансамбля из моделей типа GBM
ensemble <- h2o.stackedEnsemble(x = x,
                                y = y,
                                training_frame = train,
                                model_id = "ensemble_gbm_grid_binomial",
                                base_models = gbm_grid@model_ids)

# Оценка качества ансамбля на тестовых данных
perf <- h2o.performance(ensemble, newdata = test)

# Сравнение с качеством базовых моделей на тестовых данных
.getauc <- function(mm) h2o.auc(h2o.performance(h2o.getModel(mm),
                                                newdata = test))
baselearner_aucs <- sapply(gbm_grid@model_ids, .getauc)
baselearner_best_auc_test <- max(baselearner_aucs)
ensemble_auc_test <- h2o.auc(perf)
print(sprintf("Best Base-learner Test AUC:  %s", baselearner_best_auc_test))
print(sprintf("Ensemble Test AUC:  %s", ensemble_auc_test))

# Создание предсказаний для тестовой выборки (если требуется)
pred <- h2o.predict(ensemble, newdata = test)
```

На языке Python

```
import h2o
from h2o.estimators.random_forest import H2ORandomForestEstimator
from h2o.estimators.gbm import H2OGradientBoostingEstimator
from h2o.estimators.stackedensemble import H2OStackedEnsembleEstimator
from h2o.grid.grid_search import H2OGridSearch
from __future__ import print_function
h2o.init()

# Импорт обучающей (с бинарной переменной отклика) и тестовой выборки H2O
train = h2o.import_file("https://s3.amazonaws.com/erin-data/higgs/higgs_train_10k.csv")
test = h2o.import_file("https://s3.amazonaws.com/erin-data/higgs/higgs_test_5k.csv")

# Предикторы и целевая переменная
x = train.columns
y = "response"
x.remove(y)

# Для бинарной классификации целевая переменная должна быть фактором
train[y] = train[y].asfactor()
test[y] = test[y].asfactor()

# Количество блоков для перекрестной проверки (используются при создании «данных первого уровня»)
nfolds = 5
```

```
# Есть несколько способов получения списка моделей для стекинга:
# 1. Обучить отдельные модели и собрать их в список
# 2. Обучить несколько моделей, используя h2o.grid
# 3. Использовать h2o.grid несколько раз
# Примечание: блоки должны быть одинаковыми для всех моделей,
# также необходимо сохранить предсказания, полученные в ходе перекрестной проверки

# 1. Создание ансамбля из двух моделей (GBM + RF)

# Обучение и перекрестная проверка GBM
my_gbm = H2OGradientBoostingEstimator(distribution = "bernoulli",
                                      ntrees = 10,
                                      max_depth = 3,
                                      min_rows = 2,
                                      learn_rate = 0.2,
                                      nfolds = nfolds,
                                      fold_assignment = "Modulo",
                                      keep_cross_validation_predictions = True,
                                      seed = 1)
my_gbm.train(x = x, y = y, training_frame = train)

# Обучение и перекрестная проверка RF
my_rf = H2ORandomForestEstimator(ntrees = 50,
                                 nfolds = nfolds,
                                 fold_assignment = "Modulo",
                                 keep_cross_validation_predictions = True,
                                 seed = 1)
my_rf.train(x = x, y = y, training_frame = train)

# Обучение ансамбля из GBM и RF
ensemble = H2OStackedEnsembleEstimator(model_id = "my_ensemble_binomial",
                                       base_models = [my_gbm.model_id,
                                                      my_rf.model_id])
ensemble.train(x = x, y = y, training_frame = train)

# Оценка качества ансамбля на тестовых данных
perf_stack_test = ensemble.model_performance(test)

# Сравнение с качеством базовых моделей на тестовых данных
perf_gbm_test = my_gbm.model_performance(test)
perf_rf_test = my_rf.model_performance(test)
baselearner_best_auc_test = max(perf_gbm_test.auc(), perf_rf_test.auc())
stack_auc_test = perf_stack_test.auc()
print("Best Base-learner Test AUC:  {0}".format(baselearner_best_auc_test))
print("Ensemble Test AUC:  {0}".format(stack_auc_test))

# Создание предсказаний для тестовой выборки (если требуется)
pred = ensemble.predict(test)

# 2. Создание и стекинг моделей с использованием h2o.grid

# Гиперпараметры GBM
hyper_params = {"learn_rate": [0.01, 0.03],
                "max_depth": [3, 4, 5, 6, 9],
                "sample_rate": [0.7, 0.8, 0.9, 1.0],
```

```
                "col_sample_rate": [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]}
search_criteria = {"strategy": "RandomDiscrete", "max_models": 3, "seed": 1}

# Train the grid ***ПЕРЕВОД***
grid = H2OGridSearch(model = H2OGradientBoostingEstimator(
                                ntrees = 10,
                                seed = 1,
                                nfolds = nfolds,
                                fold_assignment = "Modulo",
                                keep_cross_validation_predictions = True),
                     hyper_params = hyper_params,
                     search_criteria = search_criteria,
                     grid_id = "gbm_grid_binomial")
grid.train(x = x, y = y, training_frame = train)

# Обучение ансамбля из моделей типа GBM
ensemble = H2OStackedEnsembleEstimator(model_id = "my_ensemble_gbm_grid_binomial",
                                       base_models = grid.model_ids)
ensemble.train(x = x, y = y, training_frame = train)

# Оценка качества ансамбля на тестовых данных
perf_stack_test = ensemble.model_performance(test)

# Сравнение с качеством базовых моделей на тестовых данных
baselearner_best_auc_test = max(
    [h2o.get_model(model).model_performance(test_data = test).auc()
    for model in grid.model_ids])
stack_auc_test = perf_stack_test.auc()
print("Best Base-learner Test AUC:  {0}".format(baselearner_best_auc_test))
print("Ensemble Test AUC:  {0}".format(stack_auc_test))

# Создание предсказаний для тестовой выборки (если требуется)
pred = ensemble.predict(test)
```

Вопросы и ответы

- **Как я могу сохранять ансамблевые модели?**

H2O поддерживает сохранение и загрузку ансамблевых моделей (см. https://0xdata.atlassian.net/browse/PUBDEV-3970). Использование этого функционала описано в разделе http://docs.h2o.ai/h2o/latest-stable/h2o-docs/save-and-load-model.html. Планируется также добавить поддержку MOJO (см. https://0xdata.atlassian.net/browse/PUBDEV-3877).

- **Всегда ли ансамбли работают лучше отдельных моделей?**

Можно надеяться, хоть это и не всегда так. Лучше выполнить сравнение качества ансамбля с качеством отдельных моделей в его составе.

- **Как я могу улучшить качество работы ансамбля?**

Если вы обнаружили, что ансамбль не превосходит отдельные модели, то у вас есть несколько вариантов действий. Прежде всего поищите в составе ансамбля базовые модели, которые работают существенно хуже остальных (например, GLM). Если таковые будут обнаружены, попробуйте построить ансамбль без них. Другой способ улучшения качества заключается в добавлении иных моделей, особенно тех, которые сильно отличаются от уже используемых. Когда будет добавлена

поддержка альтернативных алгоритмов метаобучения (https://0xdata.atlassian.net/browse/PUBDEV-3743), можно будет также проверять разные варианты этих алгоритмов.

- **Как алгоритм обрабатывает пропущенные значения?**

Обработка зависит от базовых алгоритмов. Обратитесь к документации, чтобы узнать, как они работают с пропусками.

- **Что произойдет, если целевая переменная содержит пропуски?**

Это не вызовет ошибки, строки с пропусками просто не будут использоваться.

- **Что произойдет, если вы попробуете получить предсказания для уровня категориальной переменной, которого не было в обучающей выборке?**

Это зависит от базовых алгоритмов.

- **Как алгоритм обрабатывает сильно несбалансированные (с точки зрения целевой переменной) данные?**

Используются параметры `balance_classes`, `class_sampling_factors` и `max_after_balance_size` базовых алгоритмов.

Дополнительная информация

- Презентация http://bit.ly/2uZPZKB содержит информацию о новом способе создания ансамблей, а также сравнение с пакетом **h2oEnsemble.**
- Тесты с использованием Python: http://bit.ly/2tLNlbR.
- Тесты с использованием R: http://bit.ly/2uzQoCA.

Список литературы

1. *David H. Wolpert.* Stacked Generalization // Neural Networks. Vol. 5. (1992): http://bit.ly/2uF3pco.
2. *Leo Breiman.* Stacked Regressions // Machine Learning, 24, 49–64 (1996): http://bit.ly/2u1soF0.
3. *Mark J. van der Laan, Eric C. Polley and Alan E. Hubbard.* Super Learner // Journal of the American Statistical Applications in Genetics and Molecular Biology. Vol. 6. Issue 1. (September 2007): http://bit.ly/2eMDyeE.
4. *LeDell E.* Scalable Ensemble Learning and Computationally Efficient Variance Estimation (Doctoral Dissertation). University of California. Berkeley, USA. (2015): http://bit.ly/2eQ97o1.