**Система для проведения экспериментов машинного обучения**

Предназначена для тестирования некоторых алгоритмов машинного обучения, реализованных в OpenCV (Decision Tree, Gradient Boosted Trees, Support Vector Machine, Random Trees, Extremely Randomized Trees).

# Схема использования

В одну папку требуется поместить файлы с наборами данных машинного обучения, в другую – конфигурационные файлы, в каждом из которых указаны название алгоритма, значения его параметров и, возможно, параметры кросс-валидации. Еще понадобится бенчмарк - файл, где прописаны пути к обоим папкам и указано, какие файлы в них предполагается использовать.

Чтобы провести эксперимент, нужно воспользоваться функцией из библиотеки (ее входной параметр - путь к бенчмарку, выходной - поток), которая для каждой пары набор данных - алгоритм с параметрами выполняет следующее:

1. Проводит обучение алгоритма на обучающей части набора данных. Если были указаны параметры кросс-валидации, то последняя используется для подбора параметров алгоритма.
2. Вычисляет показатели качества работы алгоритма на тестовой части набора данных.
3. По мере работы информацию о ходе эксперимента выводит в поток.

# Бенчмарк

Бенчмарк представляет собой YAML-файл со следующей структурой:

%YAML:1.0

ml\_benchmark:

data\_sets:

directory: <директория, содержащая файлы с описанием наборов данных>

config\_files:

...

-<имя файла с описанием набора данных>

...

algorithms:

directory: <директория, содержащая файлы с параметрами алгоритмов>

config\_files:

...

-<имя файла, где указан используемый алгоритм и его параметры>

...

# Задание алгоритма и его параметров

Для хранения информации об используемом алгоритме машинного обучения и его параметрах также используется YAML-файл. Его структура зависит от используемого алгоритма. Далее приведены возможные случаи.

%YAML:1.0

decision\_tree:

max\_depth: <int>

min\_sample\_count: <int>

regression\_accuracy: <float>

use\_surrogates: <bool>

max\_categories: <int>

cv\_folds: <int>

use\_1se\_rule: <bool>

truncate\_pruned\_tree: <bool>

%YAML:1.0

gradient\_boosted\_trees:

loss\_function\_type: <SQUARED\_LOSS\ABSOLUTE\_LOSS\HUBER\_LOSS\DEVIANCE\_LOSS>

weak\_count: <int>

shrinkage: <float>

subsample\_portion: <float>

max\_depth: <int>

use\_surrogates: <bool>

%YAML:1.0

support\_vector\_machine:

svm\_type: <C\_SVC\NU\_SVC\ONE\_CLASS\EPS\_SVR\NU\_SVR>

kernel\_type: <LINEAR\POLY\RBF\SIGMOID>

degree: <double>

gamma: <double

coef0: <double>

Cvalue: <double>

nu: <double>

p: <double>

term\_crit:

epsilon: <double>

max\_iter: <int>

type: <CV\_TERMCRIT\_ITER\CV\_TERMCRIT\_EPS\BOTH>

%YAML:1.0

random\_trees:

max\_depth: <int>

min\_sample\_count: <int>

regression\_accuracy: <float>

use\_surrogates: <bool>

max\_categories: <int>

calc\_var\_importance: <bool>

nactive\_vars: <int>

max\_num\_of\_trees\_in\_the\_forest: <int>

forest\_accuracy: <float>

termcrit\_type: <CV\_TERMCRIT\_ITER\CV\_TERMCRIT\_EPS\BOTH>

%YAML:1.0

extremely\_randomized\_trees:

max\_depth: <int>

min\_sample\_count: <int>

regression\_accuracy: <float>

use\_surrogates: <bool>

max\_categories: <int>

calc\_var\_importance: <bool>

nactive\_vars: <int>

max\_num\_of\_trees\_in\_the\_forest: <int>

forest\_accuracy: <float>

termcrit\_type: <CV\_TERMCRIT\_ITER\CV\_TERMCRIT\_EPS\BOTH>

Пример конфигурационного файла для алгоритма Gradient Boosted Trees:

%YAML:1.0

gradient\_boosted\_trees:

loss\_function\_type: DEVIANCE\_LOSS

weak\_count: 1000

shrinkage: 0.05

subsample\_portion: 0.8

max\_depth: 5

use\_surrogates: false

# Кросс-валидация

Если требуется использовать кросс-валидацию, нужно сделать следующие изменения в файле с параметрами алгоритма:

* в конце файла добавить строку

cv\_folds: <значение>

* для любого параметра parameter\_name, для которого требуется подобрать значение, строку

parameter\_name <значение>

нужно заменить на

parameter\_name\_grid:

min\_value: <значение>

max\_value: <значение>

step: <float>

scale: <LOGARITHMIC\LINEAR>

* Если в качестве значения scale задать LOGARITHMIC, при кросс-валидации параметр parameter\_name пробежит значения

где - наибольшее целое число, удовлетворяющее условию

На накладывается условие .

* В случае LINEAR прибавляется к , пока сумма не привысит . От требуется положительность.

Пример конфигурационного файла для Gradient Boosted Trees с параметрами кросс-валидации:

%YAML:1.0

gradient\_boosted\_trees:

loss\_function\_type: DEVIANCE\_LOSS

weak\_count: 1000

shrinkage\_grid:

min\_value: 0.005

max\_value: 0.1

step: 1.5

scale: LOGARITHMIC

subsample\_portion\_grid:

min\_value: 0.4

max\_value: 1

step: 0.05

scale: LINEAR

max\_depth: 5

use\_surrogates: false

cv\_folds: 5

# Набор данных

Для хранения набора данных используется два файла.

* Непосредственно сами данные хранятся в файле csv-подобного формата. Для его чтения система использует функцию CvMLData::read\_csv из OpenCV. Подробно об условиях, которым должен удовлетворять файл, можно узнать в документации OpenCV. Следует отметить, что большинство наборов данных из UCI Machine Learning Repository можно использовать без каких-либо изменений.
* Для хранения некоторой дополнительной информации о csv-файле с данными (номер переменной с ответами, пропорции между тестовой и обучающей выборками и др.) используется конфигурационный YAML-файл следующей структуры:

%YAML:1.0

ml\_data\_set\_header:

attributes:

...

-

name: <название переменной>

type: <тип переменной: ordered или categorical>

...

responses: <название переменной, используемой в качестве ответа>

number\_of\_samples: <общее число прецедентов>

test\_samples\_percentage: <сколько процентов от всего набора данных составляет тестовая выборка>

missed\_values: <есть ли пропущенные значения: Yes или No>

csv\_file:

filename: <имя файла с набором данных>

delimiter: <символ, используемый в файле с набором данных в качестве разделителя значений>

miss\_character: <символ, который ставится на месте пропущенного значения>

Следует отметить, что в бенчмарке указываются имена YAML-файлов и путь к директории, в которой они находятся.

Считывание csv-файла после того, как загружен YAML-файл.

1. Система пытается считать файл из пути csv\_file:filename.
2. В случае неуспеха пункта 1 делается попытка считать csv-файл из той же директории, где находится YAML-файл, с тем же именем и расширением csv или data.

# Код

Для использования библиотеки требуется подключить MLExperiments.h.

void LoadPatterns(const std::string& directory)

Загружает шаблоны из папки directory. В репозитории, где лежит код, шаблоны находятся в папке Patterns. Они используются системой для поиска ошибок в конфигурационных файлах.

void CreateMLDataHeaders(const std::string& directory,

std::ostream& stream,

const std::vector<std::string>& extentions = std::vector<std::string>::vector())

Для файлов с наборами данных создает заготовки соответствующих YAML-файлов. Вручную в них остается только указать номер переменной с ответами и (при необходимости) поправить значения некоторых параметров.

* directory – Папка, откуда берутся наборы данных. Туда же сохраняются YAML-файлы.
* stream – Поток, в который выводится информация о работе функции.
* extentions - Задает фильтр для расширений. Если вектор пуст, то перебираются все файлы; в противном случае перебираются только те файлы, расширения которых являются элементами вектора.

void Experiment(const std::string& benchmarkFilename,

std::ostream& log,

bool printCVErrors = false)

Проводит эксперимент.

* benchmarkFilename – Путь к бенчмарку.
* log – Поток, в который выводится информация о работе функции.
* printCVErrors – Если true, то в процессе кросс-валидации в log выводятся все рассматриваемые сочетания подбираемых параметров и соответствующие им валидационные ошибки.