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
  $$minValue, minValue * step, minValue * step^2, \dots, minValue * step^n,$$
  где n - наибольшее целое число, удовлетворяющее условию
  $$minValue * step^n \leq maxValue$$
  На $step$ накладывается условие $step \geq 1$.
- В случае LINEAR $step$ прибавляется к $minValue$, пока сумма не привысит $maxValue$. От $step$ 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.