Нижегородский государственный университет им. Н.И. Лобачевского

Разработка мультимедийных приложений с использованием библиотек OpenCV и IPP

Лабораторная работа

Машинное обучение в библиотеке OpenCV

Дружков П.Н., кафедра математической логики и высшей алгебры, факультет ВМК

Содержание

- Цели и задачи работы.
- Краткий обзор возможностей модуля ML библиотеки OpenCV:
  - Машина опорных векторов (SVM).
  - Дерево решений.
  - Случайный лес.
  - Градиентный бустинг деревьев решений.
- Кластеризация в библиотеке OpenCV:
  - K-means.
- Разработка приложения для решения задач классификации с помощью рассмотренных алгоритмов.
- Разработка приложения для решения задач кластеризации алгоритмом K-means.

Цели работы

- Познакомиться с базовыми возможностями модуля ML библиотеки компьютерного зрения OpenCV.
- Научиться использовать некоторые алгоритмы машинного обучения, реализованные в библиотеке OpenCV, для решения задач классификации.
- Научиться использовать алгоритм кластеризации K-means, реализованный в библиотеке OpenCV.

Задачи работы (1)

- Изучить принцип работы алгоритмов решения задачи классификации:
  - машина опорных векторов (SVM),
  - дерево решений,
  - случайный лес.
  - градиентный бустинг деревьев решений.
- Изучить принципы работы алгоритма кластеризации K-means.
- Рассмотреть прототипы функций, реализующих базовую функциональность перечисленных алгоритмов в библиотеке OpenCV.

Задачи работы (2)

- Разработать приложение для решения задач классификации с помощью рассмотренных алгоритмов.
- Изучить влияние параметров рассмотренных алгоритмов на качество обучаемой модели на основе модельных примеров.

КРАТКИЙ ОБЗОР ВОЗМОЖНОСТЕЙ МОДУЛЯ ML БИБЛИОТЕКИ OPENCV

Задачи обучения с учителем

- X – множество допустимых входов (пространство признаков).
- Y – множество допустимых выходов (целевой признак).
- Между элементами X и Y существует зависимость.
- По данным конечного набора примеров (прецедентов) $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(N)}, y^{(N)})\}$ восстановить данную зависимость.
- X может иметь сложную структуру, будем рассматривать $X = \mathbb{R}^d$.
- $Y = \mathbb{R}$ – задача восстановления регрессии.
- $Y = \{1,2, \dots, K\}$ – задача **классификации**.

Алгоритмы обучения с учителем

- Нормальный байесов классификатор
- Метод k ближайших соседей
- Искусственная нейронная сеть
- **Машина опорных векторов (SVM)**
- **Дерево решений (алгоритм CART)**
- Бустинг
  - Adaboost (Discrete, Real, LogitBoost, Gentle)
  - **Градиентный бустинг**
- Бэггинг
  - **Случайный лес**
  - Экстремально случайный лес

Машина опорных векторов (SVM). Алгоритм (1)

- Идея алгоритма машины опорных векторов (Support Vector Machine, SVM) заключается в построении оптимальной поверхности $\beta h(x) + \beta_0 = 0$, представляющей собой гиперплоскость в спрямляющем пространстве $\mathcal{H} = h(\mathcal{X})$, разделяющей точки $x^{(i)}$ различных классов из обучающей выборки.
- При этом решается оптимизационная задача

$$\min_{\beta,\beta_0,\xi} \frac{1}{2}\|\beta\| + C\sum_{i=1}^{N} \xi_i$$

$$y^{(i)}\left(\beta h\left(x^{(i)}\right) + \beta_0\right) \geq 1 - \xi_i, \quad \xi_i \geq 0, \qquad i = \overline{1, N}.$$

Машина опорных векторов (SVM). Алгоритм (2)

- Отображение h задается неявно с помощью ядра

$$K(x, x') = h(x)h(x')$$

- Решающая функция записывается в виде

$$f(x) = \text{sign}(h(x)\beta + \beta_0) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y^{(i)} K\left(x^{(i)}, x\right) + \beta_0\right)$$

- Функция зависит от некоторых точек $x^{(i)}$ из обучающей выборки, называемых опорными векторами.

SVM. Ядра (1)

- Линейное

$$K(x, x') = \langle x, x' \rangle$$

- Полиномиальное

$$K(x, x') = (\gamma \langle x, x' \rangle + coef_0)^{degree}$$

- Гауссово или радиальная функция

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$

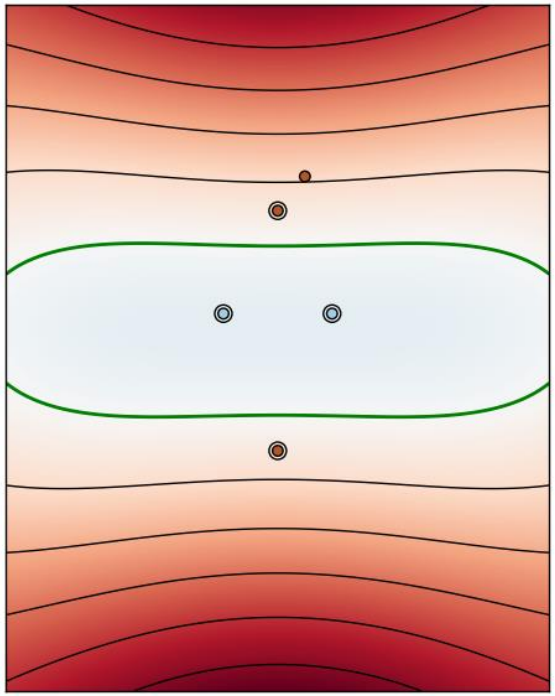
- Сигмоидальное

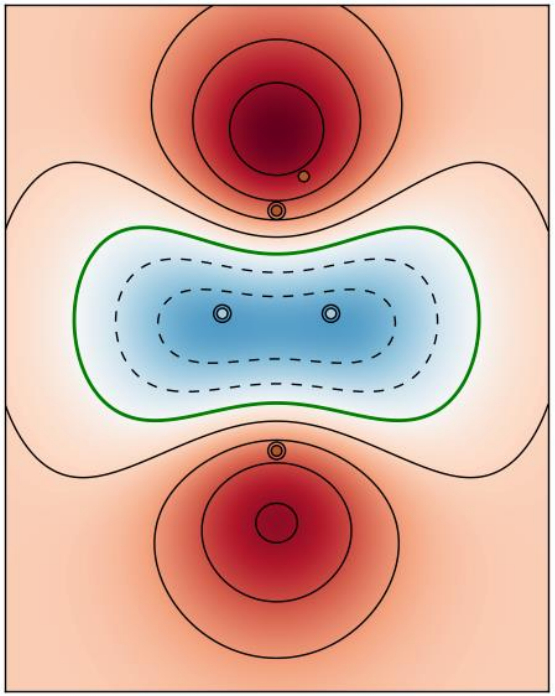
$$K(x, x') = \tanh(\gamma \langle x, x' \rangle + coef_0)$$

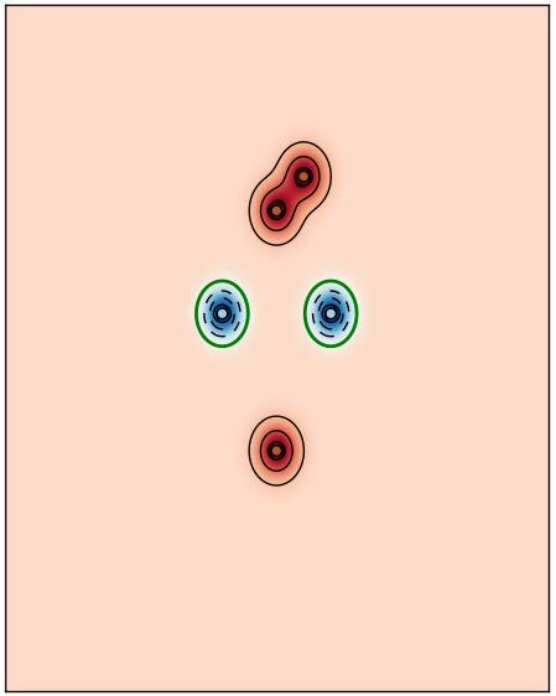
- …

SVM. Ядра (2)

- Для получения модели с хорошей обобщающей способностью важно не только выбрать подходящее ядро, но и подобрать его параметры.







CvSVMParams

- Для представления параметров SVM в OpenCV используется структура `CvSVMParams`

```
struct CvSVMParams
{
    int             svm_type;
    int             kernel_type;
    double          degree;
    double          gamma;
    double          coef0;
    double          C;
    CvTermCriteria term_crit;
    ...
};
```

CvSVM (1)

- Функционал по обучению SVM и ее использованию для предсказаний содержится в классе `CvSVM`:

```
class CvSVM : public CvStatModel
```

- Класс содержит перечисления для задания используемой модификации алгоритма SVM и ядер:

```
enum { C_SVC=100, NU_SVC=101, ONE_CLASS=102,
EPS_SVR=103, NU_SVR=104 };

enum { LINEAR=0, POLY=1, RBF=2, SIGMOID=3 };
```

CvSVM (2)

- Обучение модели производится с помощью метода `train`

```
bool train(const Mat& trainData,
           const Mat& responses,
           const Mat& varIdx=Mat(),
           const Mat& sampleIdx=Mat(),
           CvSVMParams params=CvSVMParams());
```

- `trainData` – матрица входных признаков, построчно содержащая $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$.
- `responses` – матрица значений целевого признака.
- `varIdx` – маска используемых признаков.
- `sampleIdx` – маска или перечисление индексов используемых прецедентов обучающей выборки.
- `params` – параметры SVM.

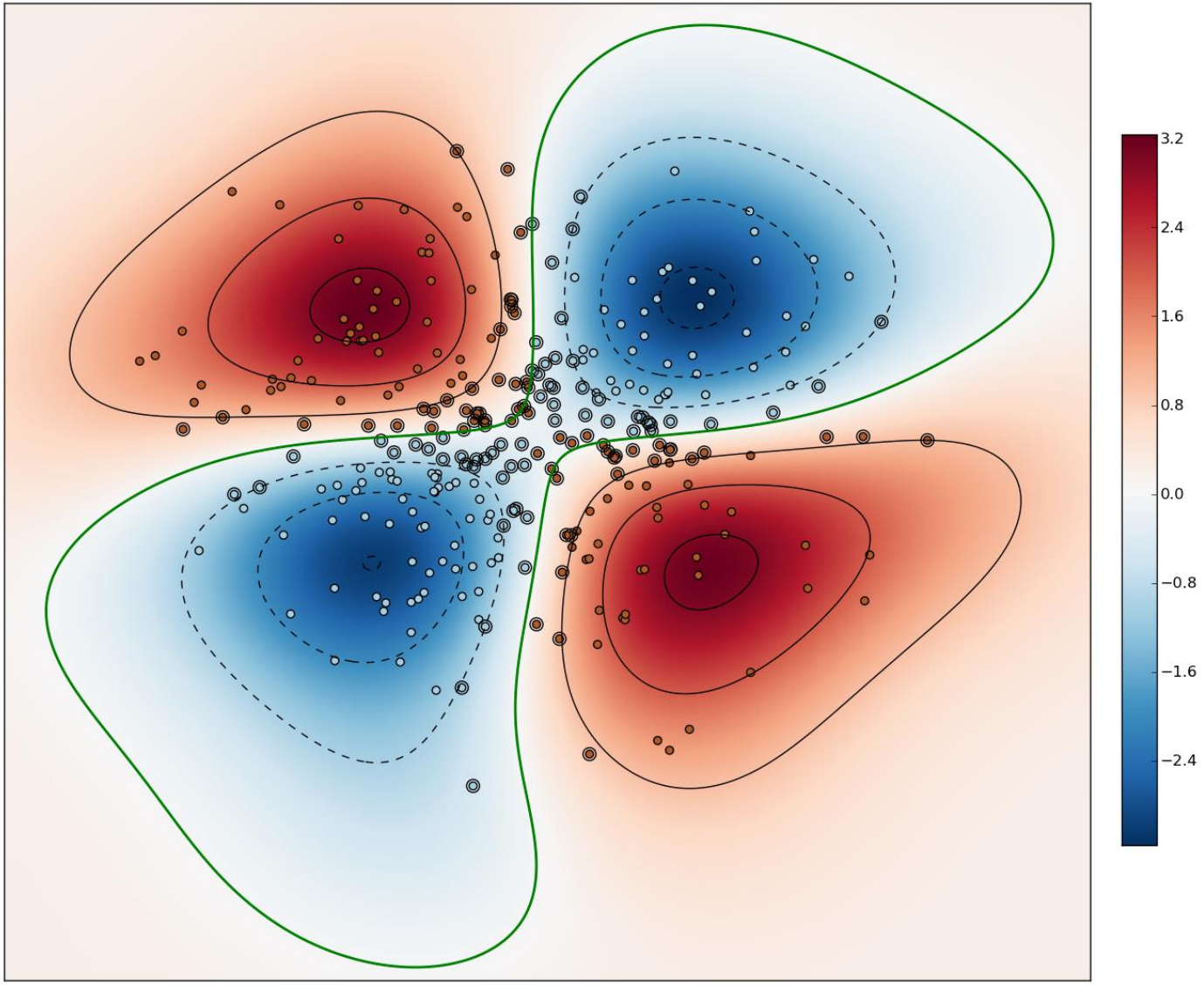
CvSVM (3)

- Предсказание SVM-модели производится с помощью метода `predict`

```
float predict(const Mat& sample,
              bool returnDFVal=false) const;
```

- `sample` – матрица-строка, содержащая признаки классифицируемой точки x.
- `returnDFVal` – контролирует тип возвращаемого значения, если `returnDFVal=true` и решается задача бинарной классификации, то возвращается значение функции $\sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + \beta_0$, иначе возвращается номер класса.

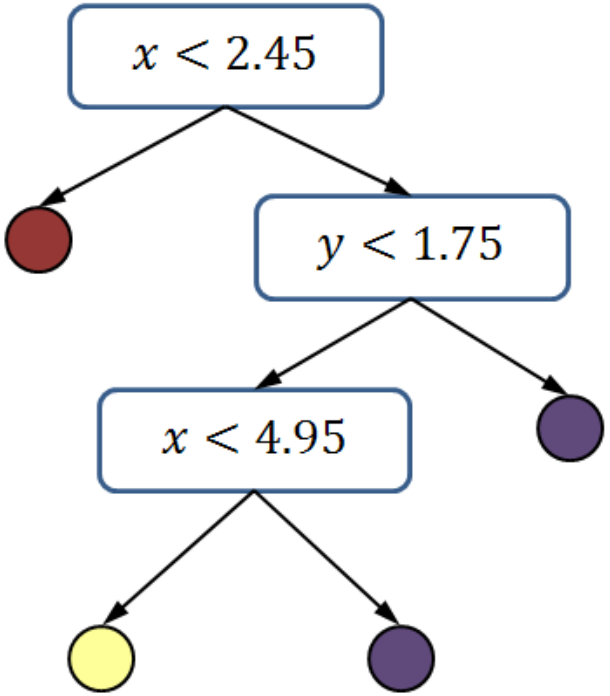
SVM. Пример

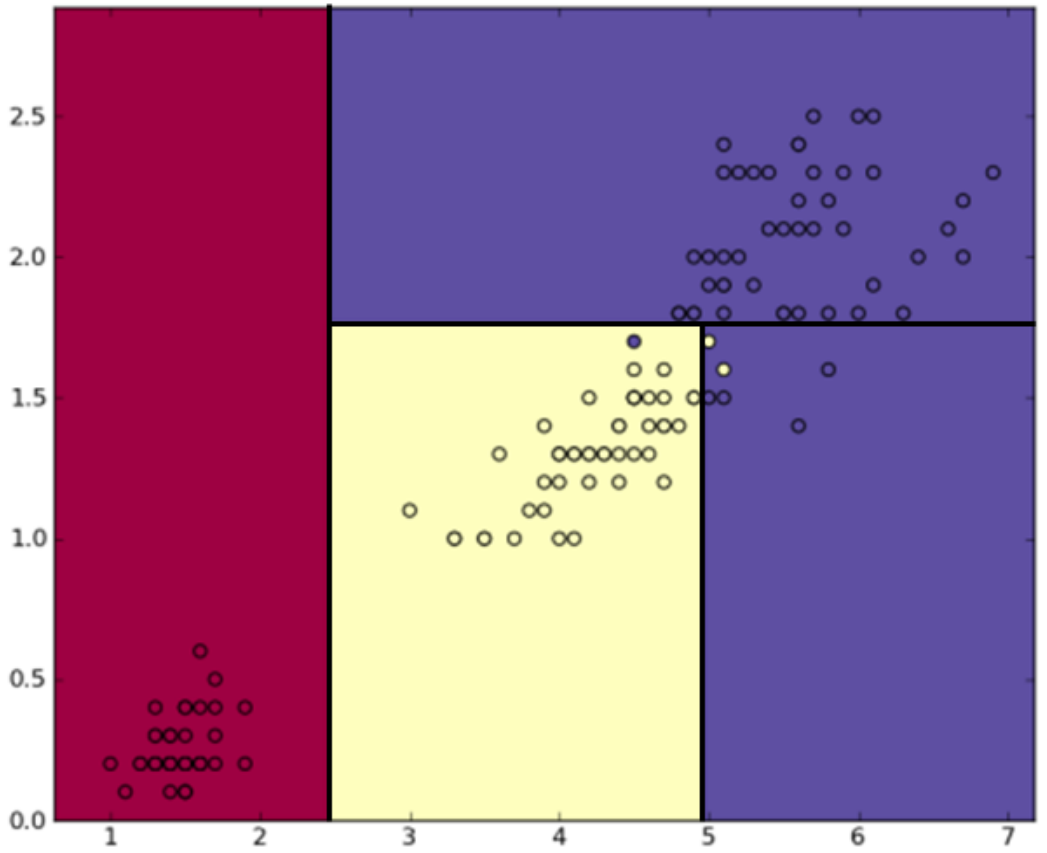


Дерево решений (1)

- Пространство признаков разбивается на параллелепипеды со сторонами, параллельными осям координат.
- Разбиение описывается в виде дерева.
- Каждому узлу дерева соответствует область пространства признаков (в виде параллелепипеда).
- Внутренним узлам также присвоено правило разбиения, соответствующих им областей. Листовым узлам – метка класса, присваиваемая всем точкам данной области.

Дерево решений (2)





Дерево решений. Алгоритм обучения

- Используются жадные алгоритмы построения дерева решений.
- В каждом узле выполняется поиск разбиения, максимально уменьшающего неоднородность (загрязнение, impurity) точек обучающей выборки в соответствующей области.
- Разбиение производится до тех пор, пока в вершины попадает достаточное количество точек обучающей выборки (больше заданного значения), или до тех пор, пока не будет построено дерево заданной высоты (глубины).
- С целью борьбы с переобучением, после построения применяют процедуру отсечения (pruning) поддеревьев.

CvDTreeParams

- Для представления параметров дерева решений в OpenCV используется структура `CvDTreeParams`

```
struct CvDTreeParams
{
    int    max_depth;
    int    min_sample_count;
    int    cv_folds;
    ...
};
```

CvDTree (1)

- Функционал по обучению дерева решений и его использованию для предсказаний содержится в классе `CvDTree`

```
class CvDTree : public CvStatModel
```

CvDTree (2)

- Обучение модели производится с помощью метода `train`

```
bool train(const Mat& trainData,
           int tflag,
           const Mat& responses,
           const Mat& varIdx=Mat(),
           const Mat& sampleIdx=Mat(),
           const Mat& varType=Mat(),
           const Mat& missingDataMask=Mat(),
           CvDTreeParams params=CvDTreeParams());
```

- `trainData` – матрица входных признаков.
- `tflag` – принимает значение `CV_ROW_SAMPLE`, если точки $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$ соответствуют строкам матрицы `trainData`, `CV_COL_SAMPLE` – если столбцам.
- `responses` – матрица значений целевого признака.

CvDTree (3)

- `varIdx` – маска используемых признаков.
- `sampleIdx` – маска или перечисление индексов используемых прецедентов обучающей выборки.
- `varType` – матрица размера $1\text{x}(d+1)$, задающая типы признаков: первые d элементов отвечают за тип входных признаков, последний – тип целевой переменной. Для удобства в OpenCV определены константы `CV_VAR_ORDERED` и `CV_VAR_CATEGORICAL`.
- `missingDataMask` – маска пропущенных значений
- `params` – параметры алгоритма обучения.

- По умолчанию целевой признак считается категориальным (решается задача классификации), все остальные признаки – количественные.

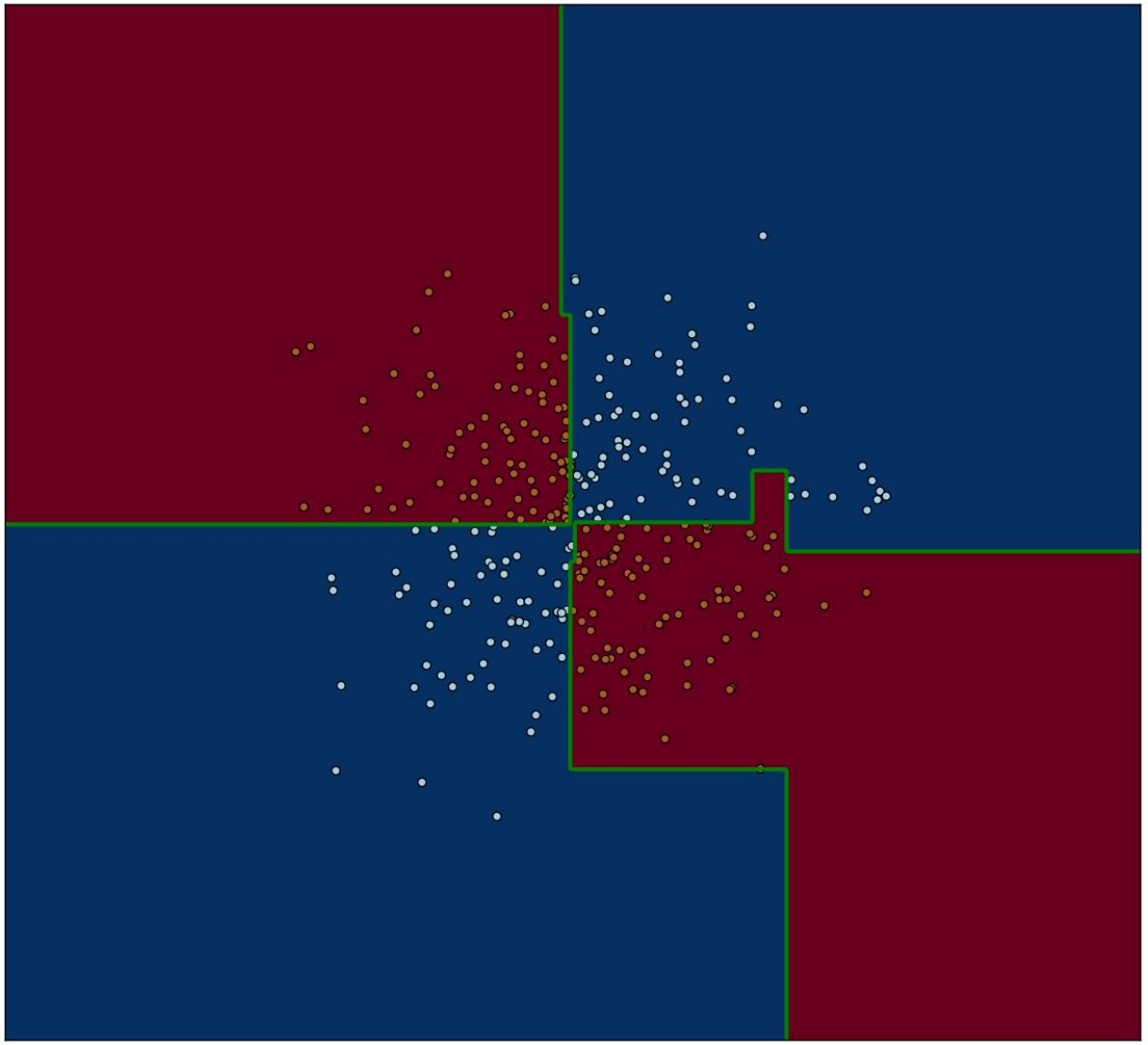
CvDTree (4)

- Предсказание производится с помощью метода `predict`

```
CvDTreeNode* predict(const Mat& sample,
                     const Mat& missingDataMask=Mat(),
                     bool preprocessedInput=false) const;
```

- `sample` – матрица-строка, содержащая признаки классифицируемой точки x.
- `missingDataMask` – маска пропущенных значений.
- `preprocessedInput` – определяет порядок работы с категориальными признаками в ходе предсказания.
- Функция возвращает указатель на структуру, описывающую лист дерева. Предсказанное значение целевого признака находится в поле `value`.

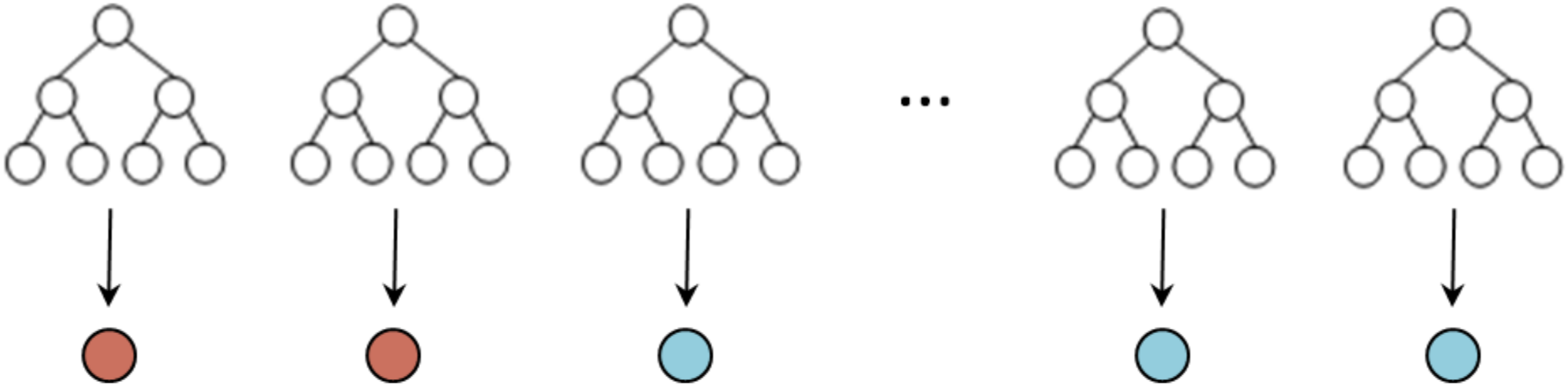
Дерево решений. Пример

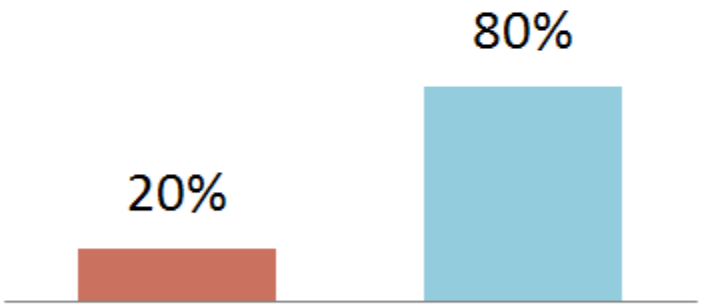


Случайный лес (1)

- Один из алгоритмов, основанных на группировке моделей (обучении ансамбля).
- Относится к категории bagging-алгоритмов, т.е. пытается строить много независимых моделей.
- Обучается заданное количество деревьев решений.
- Каждое дерево решений строится на модифицированной обучающей выборке: по схеме выбора с возвращением случайным образом генерируется новая выборка.
- При построении каждого узла дерева используются не все признаки: случайным образом выбирается заданное количество признаков.
- Отсечение (pruning) деревьев решений не осуществляется.

Случайный лес (2)





Классификация производится на основе голосования деревьев решений.

CvRTParams

- Для представления параметров случайного леса в OpenCV используется структура `CvRTParams`

```
struct CvRTParams : public CvDTreeParams
{
    int nactive_vars;
    CvTermCriteria term_crit;
    ...
};
```

CvRTrees (1)

- Функционал по обучению случайного леса и его использованию для предсказаний содержится в классе `CvRTrees`

```
class CvRTrees : public CvStatModel
```

CvRTrees (2)

- Обучение модели производится с помощью метода train

```
bool train(const Mat& trainData,
           int tflag,
           const Mat& responses,
           const Mat& varIdx=Mat(),
           const Mat& sampleIdx=Mat(),
           const Mat& varType=Mat(),
           const Mat& missingDataMask=Mat(),
           CvRTParams params=CvRTParams());
```

- Параметры метода полностью аналогичны параметрам `CvDTree::train`.

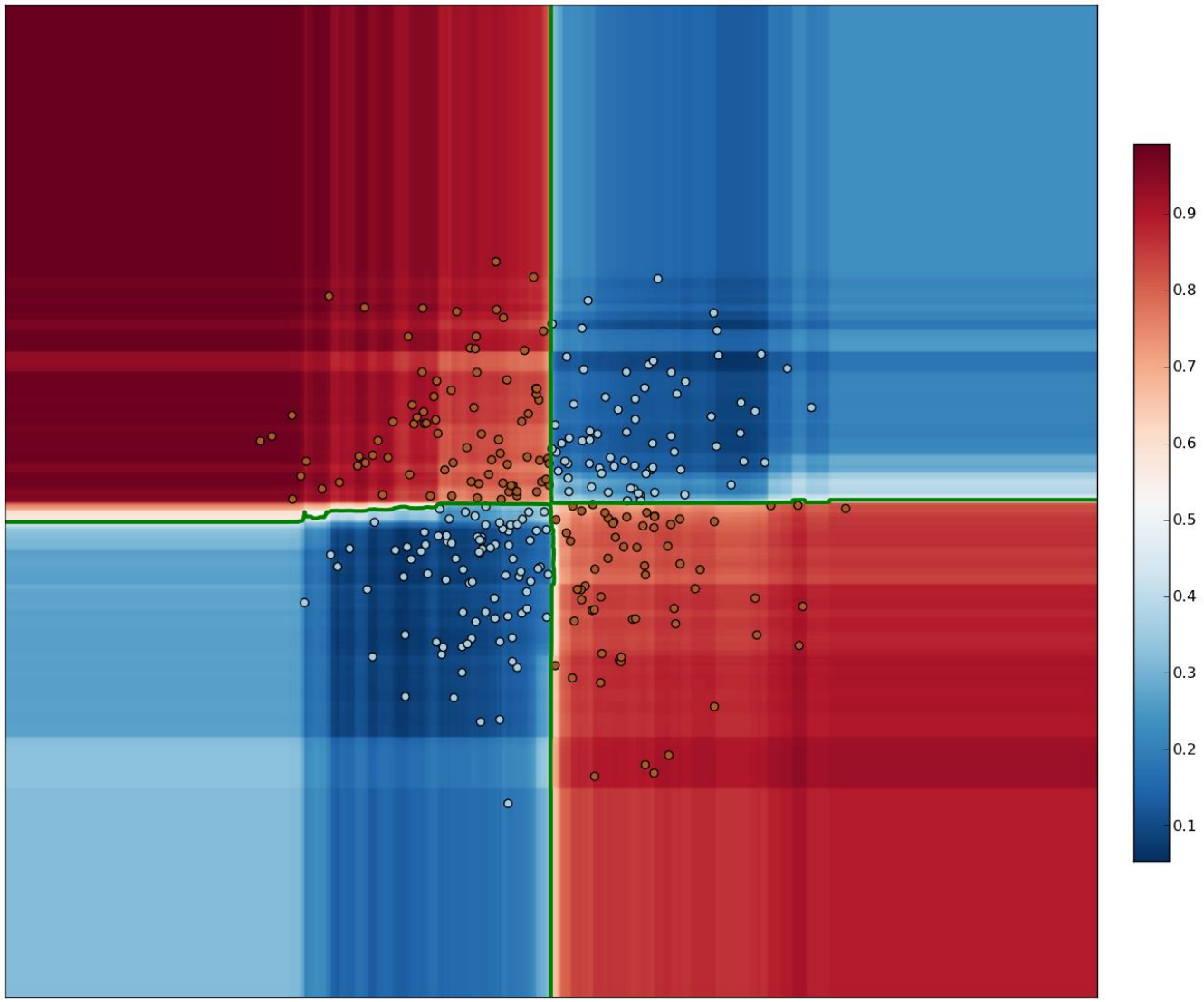
CvRTrees (3)

- Предсказание производится с помощью метода `predict`

```
float predict(const Mat& sample,
              const Mat& missing = Mat()) const;
```

- `sample` – матрица-строка, содержащая признаки классифицируемой точки x
- `missing` – маска пропущенных значений
- Функция возвращает предсказанное значение.

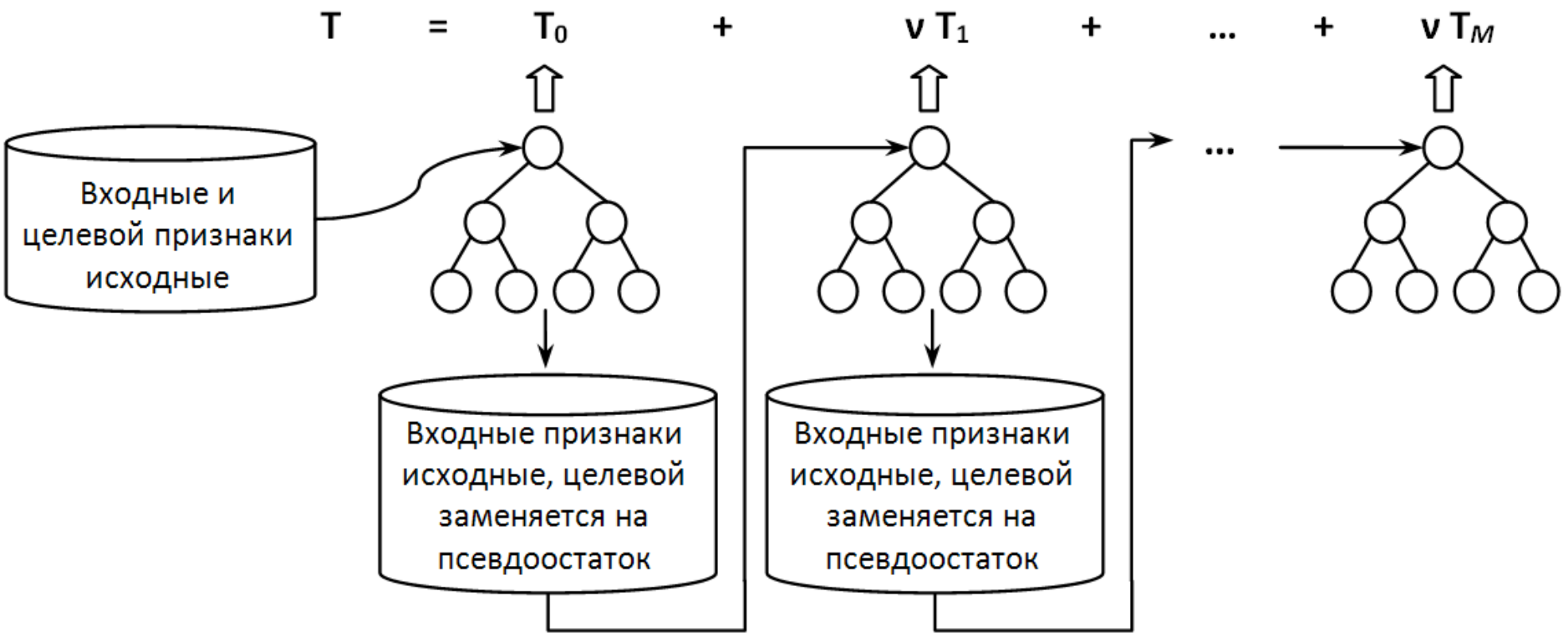
Случайный лес. Пример



Градиентный бустинг деревьев решений (1)

- Строит модель в виде суммы базовых (слабых) моделей.
- В качестве базовых моделей используются деревья решений.
- Новые слагаемые-деревья добавляются в сумму из соображений жадной минимизации эмпирического риска, заданного некоторой функцией потерь $L(y, y') = L\big(y, f(x)\big)$.
- Алгоритм без серьезной модификации может применяться для любой дифференцируемой функции потерь.
- Как правило, используются деревья небольшой высоты, отсечение (pruning) не осуществляется.

Градиентный бустинг деревьев решений (2)



CvGBTreesParams (1)

- Для представления параметров градиентного бустинга деревьев решений в OpenCV используется структура `CvGBTreesParams`

```
struct CvGBTreesParams : public CvDTreeParams
{
    int weak_count;
    int loss_function_type;
    float subsample_portion;
    float shrinkage;
    ...
};
```

CvGBTreesParams (2)

- `loss_function_type` – тип используемой функции потерь. Список реализованных штрафных функций представлен перечислением в классе `CvGBTrees`:

```
enum {SQUARED_LOSS=0, ABSOLUTE_LOSS,
      HUBER_LOSS=3, DEVIANCE_LOSS};
```

  `SQUARED_LOSS`, `ABSOLUTE_LOSS`, `HUBER_LOSS` предназначены для задач восстановления регрессии, `DEVIANCE_LOSS` – для задач классификации.

- `weak_count` – количество обучаемых деревьев.
- `shrinkage` – скорость обучения.
- `subsample_portion` – доля выборки, используемая для обучения каждого дерева.

CvGBTrees (1)

- Функционал по обучению модели градиентного бустинга деревьев решений и ее использованию для предсказаний содержится в классе `CvGBTrees`

```
class CvGBTrees : public CvStatModel
```

CvGBTrees (2)

- Обучение модели производится с помощью метода train

```
bool train( const Mat& trainData,
            int tflag,
            const Mat& responses,
            const Mat& varIdx=Mat(),
            const Mat& sampleIdx=Mat(),
            const Mat& varType=Mat(),
            const Mat& missingDataMask=Mat(),
            CvGBTreesParams params=CvGBTreesParams(),
            bool update=false );
```

- Параметры метода аналогичны параметрам `CvDTree::train` и `CvRTrees::train`. Параметр `update` фиктивный.

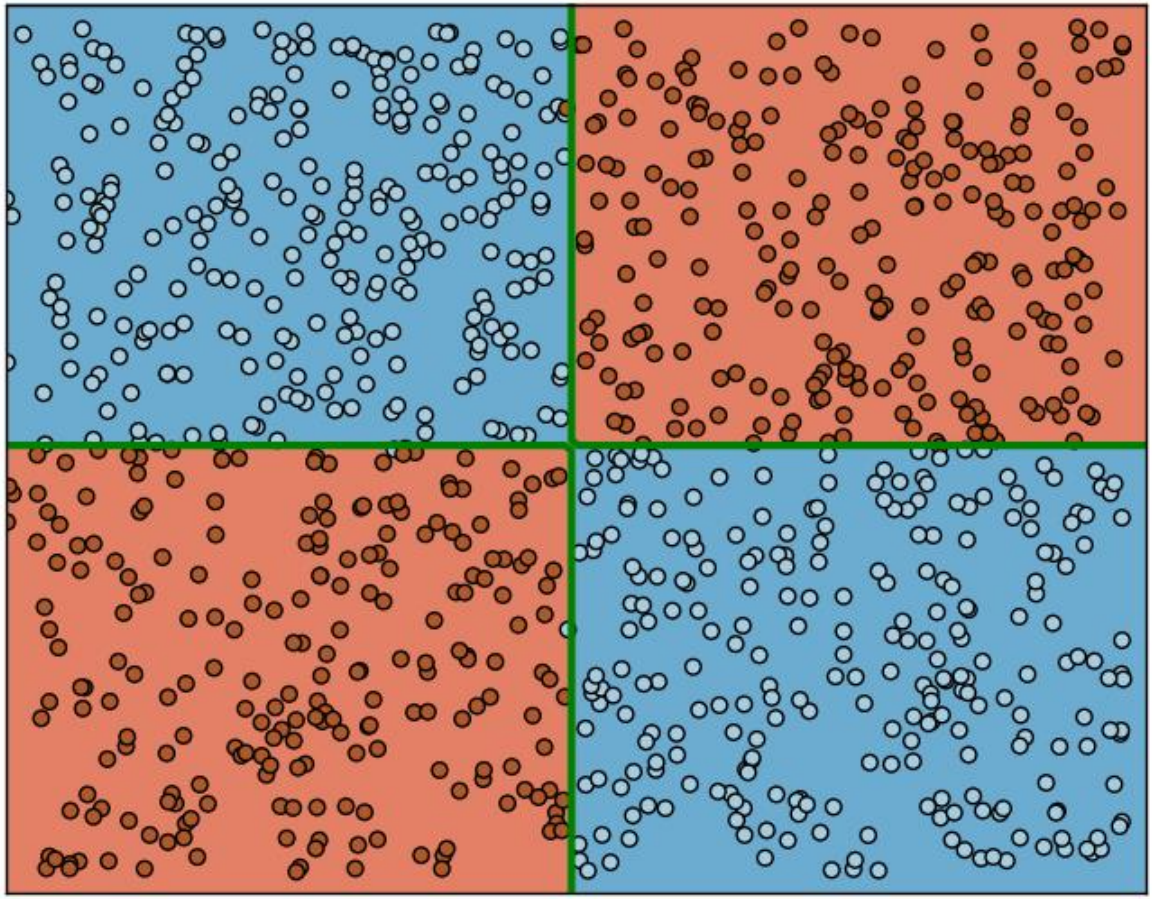
CvGBTrees (3)

- Предсказание производится с помощью метода `predict`

```
float predict(const Mat& sample,
        const Mat& missing=Mat(),
        const Range& slice=Range::all(),
        int k=-1) const;
```

- `sample` – матрица-строка, содержащая признаки классифицируемой точки x.
- `missing` – маска пропущенных значений.
- `slice` – диапазон используемых деревьев.
- `k` – актуален только для функции потерь `DEVIANCE_LOSS`, если $0 \le k < K$, то функция возвращает сумму деревьев для *k*-го класса.
- Функция возвращает предсказанное значение.

Градиентный бустинг деревьев решений. Пример



КЛАСТЕРИЗАЦИЯ В БИБЛИОТЕКЕ OPENCV

Задача кластеризации

- Разбить выборку объектов, заданных с помощью их признакового описания, на подмножества (кластеры).
- Объекты внутри одного кластера схожи.
- Объекты разных кластеров существенно отличаются.
- Одним из способов формализации понятия похожести является использование расстояния между точками в пространстве признаков, т.е. близко расположенные точки похожи, далеко – нет.

K-means (1)

- Пусть выборка уже разбита на K частей, т.е. каждому объекту x_i поставлен в соответствие номер кластера $C(i)$.
- Для каждого вычисляется центр масс

$$m_t = \frac{\sum_{\mathrm{C}(i)=t} x_i}{|\{i\colon \mathrm{C}(i) = t\}|}$$

- Выполняется минимизация суммы квадратов расстояний всех точек до центров соответствующих им кластеров

$$\min_{C,m} \sum_{i=1}^{N} \left\| x_i - m_{C(i)} \right\|^2$$

K-means (2)

- Поочередно выполняем минимизацию по m и C :
  - Для заданного разбиения вычисляем центры кластеров.
  - Для заданных центров разбиваем точки на кластеры.
- Проблемы:
  - Не гарантируем нахождение глобального минимума.
  - Как выбирать начальное разбиение?
  - Как выбрать число кластеров?

Функция kmeans (1)

```
double kmeans(InputArray data,
              int K,
              InputOutputArray bestLabels,
              TermCriteria criteria,
              int attempts,
              int flags,
              OutputArray centers=noArray())
```

- `data` – матрица с выборкой. Каждая строка соответствует объекту выборки.
- `K` – количество кластеров.
- `bestLabels` – матрица, содержащая для каждого объекта выборки номер кластера, к которому он отнесен.
- `criteria` – критерий остановки итерационной процедуры. По количеству итераций или по величине смещения центров кластеров.

Функция kmeans (2)

- `attempts` – количество запусков алгоритма с использованием различных начальных разбиений.
- `flags` – определяет вид начального разбиения:
  - `KMEANS_RANDOM_CENTERS` – центры генерируются случайным образом;
  - `KMEANS_USE_INITIAL_LABELS` – используется заданное с помощью параметра `centers` начальное разбиение;
  - `KMEANS_PP_CENTERS` – центры генерируются с помощью алгоритм k-means++.
- `centers` – матрицы центров кластеров.

- Функция возвращает наилучший из полученных показателей компактности $\sum_{i=1}^{N} \left\| x_i - m_{C(i)} \right\|^2$.

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧИ КЛАССИФИКАЦИИ

Требования к приложению

- Решение задачи классификации с помощью SVM, дерева решений, случайного леса и градиентного бустинга деревьев решений.
- Обучение выбранной модели с заданными параметрами.
- Вычисление ошибки классификации на обучающей и тестовой выборках.

Структура приложения

- В предоставленном проекте уже реализованы функции:
  - чтения наборов данных из YAML-файлов;
  - графического отображения результатов работы алгоритмов в случае двумерного пространства признаков;
  - общая логика обучения, тестирования и визуализации.

Задания (1)

- Реализуйте функции, вызывающие алгоритмы обучения и предсказания:
  - `trainSVM` и `getSVMPrediction` (файл cvsvm.cpp);
  - `trainDTree` и `getDTreePrediction` (файл cvdtree.cpp);
  - `trainRTrees` и `getRTreesPrediction` (файл cvrtrees.cpp);
  - `trainGBTrees` и `getGBTreesPrediction` (файл cvgbtrees.cpp)
- Реализуйте функцию вычисления ошибки классификации на некоторой выборке:
  - `getClassificationError` (файл main.cpp).

Задания (2)

- Примените рассмотренные алгоритмы с различными параметрами к модельным данным (уже разбиты на обучающую и тестовую части):
  - dataset1.yml,
  - dataset2.yml,
  - dataset3.yml,
  - dataset4.yml,
  - datasetMulticlass.yml,
  - datasetHighDim.yml.
- *Для SVM реализуйте функцию `getSupportVectors` (файл cvsvm.cpp) для получения матрицы из опорных векторов по обученной модели. Используйте методы `get_support_vector_count` и `get_support_vector` класса `CvSVM`.

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧИ КЛАСТЕРИЗАЦИИ МЕТОДОМ K-MEANS

Структура приложения

- Переменные:
  - `points` – матрица объектов выборки.
  - `labels` – номера кластеров объектов выборки.
  - `centers` – матрица полученных центров кластеров.
  - `K` – количество кластеров.
- В предоставленном проекте уже реализованы функции визуализации.

Задания

- Напишите вызов функции `kmeans` в файле main.cpp.
- Примените рассмотренные алгоритмы с различными параметрами к модельным данным:
  - dataset1.yml,
  - dataset2.yml,
  - dataset3.yml,
  - dataset4.yml.
- Проанализируйте полученные результаты.

Авторский коллектив

- Дружков Павел Николаевич,
  аспирант кафедры математической логики и высшей алгебры факультета ВМК ННГУ

  druzhkov.paul@gmail.com