DiabetesAI

1.0.0

Создано системой Doxygen 1.9.7

Глава 1

Алфавитный указатель классов

1.1 Классы

Классы с их кратким описанием.

Глава 2

Список файлов

2.1 Файлы

Полный список документированных файлов.

Глава 3

Классы

3.1 Класс DiabetesData

Класс DiabetesData предназначен для подготовки датасета

#include <diabetes_data.hpp>

Открытые члены

- DiabetesData ()=default
- DiabetesData (const string &file_path, const string &data_name)
- DiabetesData (const vector< vector< double > > &features)
- void load_data_from_file (const string &file_path, const string &data_name)
- vector< vector< double > > get_X ()
- vector< int > get_y ()

Открытые статические члены

- static vector< vector< double > > data_normalization (vector< vector< double > > X)

Закрытые данные

- vector< vector< double > > features_
- vector< vector< string > > dataset_
- vector< vector< double > > X_
- vector< int > y_

3.1.1 Подробное описание

Класс DiabetesData предназначен для подготовки датасета

Класс DiabetesData предоставляет возможность подготовить датасет для дальнейшей работы с классом LogisticRegression и Plot, путем разбиения датасета на двумерный вектор double и одномерный вектор int: вектор X_ содержит в себе строки датасета, не включая названия столбцов и исход(outcome), и y_, содержит в себе лишь исход(outcome)

3.1.2 Конструктор(ы)

3.1.2.1 DiabetesData() [1/3]

```
DiabetesData::DiabetesData ( )  [default]
```

Конструктор по умолчанию для создания объекта класса

3.1.2.2 DiabetesData() [2/3]

```
DiabetesData::DiabetesData (
            const string & file_path,
            const string & data_name )  [explicit]
```

Конструктор принимает на вход полный путь к директории с датасетами и название датасета, после чего он вызывает функцию load_data_from_file

Аргументы

in	file_path	Полный путь к репозиторию с датасетами вида: ¨C:\\...\\data¨
in	data_name	Название файла .csv с датасетои вида: ¨dataset¨

```
load_data_from_file(file_path, data_name);
```

3.1.2.3 DiabetesData() [3/3]

```
DiabetesData::DiabetesData (
            const vector< vector< double > > & features )  [explicit]
```

Конструктор принимает на вход выборку и нормализует её с помощью data_normalization

Аргументы

in	features	двумерный вектор double состоящий из одной строчки и n столбцов (n = кол-ву столбцов датасета - 1)

```
if (features.size() != dataset_[0].size() || features.empty())
    throw runtime_error(
          "The number of parameters does not match the number of dataset parameters, check the correctness of the entered data
       and check which dataset you submitted");
features_ = data_normalization(features);
```

3.1.3 Методы

3.1.3.1 data_normalization()

```
vector< vector< double > > DiabetesData::data_normalization (
            vector< vector< double > > X )  [static]
```

Функция принимает на вход выборку X_ и нормализует её методом z-масштабирования

Аргументы

X	Двумерный вектор типа double, содержащий в себе выборку

```
vector<double> avarage;
for (int j = 0; j < X[0].size(); j++) {
    double summ = 0;
    for (auto &i: X) {
        summ += i[j];
    }
    avarage.push_back(summ / X.size());
}

vector<double> deviation;
for (int j = 0; j < X[0].size(); j++) {
    double summ = 0;
    for (auto &i: X)
        summ += pow((i[j] - avarage[j]), 2);

    deviation.push_back(sqrt(summ / (X.size() - 1)));
}

for (int j = 0; j < X[0].size(); j++) {
    for (auto &i: X) {
        i[j] = (i[j] - avarage[j]) / deviation[j];
        if (i[j] > 3 * deviation[j]) {
            i[j] = 0;
        }
    }
}
return X;
```

3.1.3.2 get_X()

vector< vector< double > > DiabetesData::get_X ( )

Функция возвращающая приватный член X_

Возвращает

двумерный вектор типа double

```
return X_;
```

3.1.3.3 get_y()

vector< int > DiabetesData::get_y ( )

Функция возвращающая приватный член y_

Возвращает

вектор типа int

```
return y_;
```

3.1.3.4 load_data_from_file()

void DiabetesData::load_data_from_file (
const string & file_path,
const string & data_name)

Функция принимает на вход полный путь к директории с датасетами и название датасета, обрабатывает его, и записывает результат в X_ и y_

Аргументы

file_path	Полный путь к репозиторию с датасетами вида: ¨C:\\...\\data¨
data_name	Название файла .csv с датасетои вида: ¨dataset¨

```
fstream fin;
 fin.open(file_path + "\\" + data_name + ".csv", ios::in);
 string line;
 vector<vector<string» parsedCsv;
 if (fin.fail()) {
     cout « "NOT OPEN";
 }
 while (getline(fin, line)) {
     stringstream lineStream(line);
     string cell;
     vector<string> parsedRow;
     while (getline(lineStream, cell, ',')) {
         parsedRow.push_back(cell);
     }

     parsedCsv.push_back(parsedRow);
 }
 fin.close();
 dataset_ = parsedCsv;

 for (int i = 1; i < dataset_.size(); i++) {
     vector<double> cell;
     for (int j = 0; j < dataset_[1].size(); j++) {
         if (j != dataset_[1].size() - 1) {
             cell.push_back(stod(dataset_[i][j]));
         } else if (j == dataset_[1].size() - 1) {
             y_.push_back(stoi(dataset_[i][j]));
         }
     }
     X_.push_back(cell);
 }
 X_ = data_normalization(X_);
```

3.1.4 Данные класса

3.1.4.1 dataset_

vector<vector<string> > DiabetesData::dataset_ [private]

Двумерный вектор типа double, содержащий в себе выборку

3.1.4.2 features_

vector<vector<double> > DiabetesData::features_ [private]

Двумерный вектор типа double, содержащий в себе выборку для использования на уже обученной модели

3.1.4.3 X_

vector<vector<double> > DiabetesData::X_ [private]

Двумерный вектор типа double, содержащий в себе нормализованную выборку

3.1.4.4 y_

vector<int> DiabetesData::y_ [private]

Вектор типа int, содержащий в себе итог(outcome)

Объявления и описания членов классов находятся в файлах:

- prj.lab/diabetes_data/include/diabetes_data/diabetes_data.hpp
- prj.lab/diabetes_data/diabetes_data.cpp

3.2 Класс LogisticRegression

Класс LogisticRegression предназначен для обучения модели на основе лог.регрессии

#include <regression.hpp>

Открытые члены

- LogisticRegression (const vector< vector< double > > &X, const vector< int > &y)
- vector< double > fit (int max_iter=100, double lr=0.1)
- double loss (vector< int > y, vector< vector< double > > z)

Открытые статические члены

- static vector< vector< double > > logit (vector< vector< double > > X, vector< double > w)
- static vector< vector< double > > sigmoid (vector< vector< double > > logits)
- static void save_weights (const vector< double > &weights)
- static vector< vector< double > > predict_proba (vector< vector< double > > feauters)
- static vector< int > predict (const vector< vector< double > > &feauters, double threshold=0.5)
- static int model_accuracy (vector< int > results, vector< int > y)
- static void saveLossToCSV (const vector< double > &losses)

Закрытые данные

- vector< vector< double > > X_
- vector< int > y_
- vector< double > w_
- vector< double > losses_
- int max_iter_ = 0
- double lr_ = 0

3.2.1 Подробное описание

Класс LogisticRegression предназначен для обучения модели на основе лог.регрессии

Класс LogisticRegression предоставляет возможность обучить модель, основанную на лог.регрессии, путем градиентного спуска по функции потерь с использованием R2 - регуляризации

3.2.2 Конструктор(ы)

3.2.2.1 LogisticRegression()

```
LogisticRegression::LogisticRegression (
            const vector< vector< double > > & X,
            const vector< int > & y )
```

Конструктор сохраняет поступившие данные и рандомит веса для дальнейшей работы

Аргументы

X	Двумерный вектор типа double, содержащий в себе нормализованную выборку
y	Вектор типа int, содержащий в себе итог(outcome)

```
X_ = X;
y_ = y;
for (int i = 0; i < X[0].size() + 1; i++) {
    double a = rand() % 1000;
    w_.push_back(a / 1000);
}
```

3.2.3 Методы

3.2.3.1 fit()

```
vector< double > LogisticRegression::fit (
                int max_iter = 100,
                double lr = 0.1 )
```

Функция, обучающая нашу модель по градиенту функции потерь логистической регрессии

Аргументы

max_iter	Переменная типа int, содержащая в себе кол-во итераций, которые будет совершать модель при обучении

Предупреждения

Большие значение дают большую точность, но значительно увеличивают время обучения, для быстрого результата советуем использовать значение 10

Аргументы

lr	Переменная типа double, содержащая в себе коэффициент скорости обучения модели

Возвращает

Вектор типа double, содержащий в себе результаты функции потерь на каждой итерации модели при обучении

3.2.3.2 logit()

```
vector< vector< double > > LogisticRegression::logit (
                vector< vector< double > > X,
                vector< double > w )   [static]
```

Функция подсчитывает логиты всей выборки по текущим весам и возвращает их

Аргументы

X	Транспонированный двумерный вектор типа double, содержащий в себе выборку
w	Вектор весов типа double

Возвращает

Двумерный вектор логитов типа double

Формула по который мы вычисляем логиты:

$$\sum_{i=0}^{n} b + w_i * x_i$$

```
unsigned long long int m = X.size();
unsigned long long int n = X[0].size();
vector<vector<double» logits(m, vector<double>(1, 0.0));
for (int i = 0; i < m; i++) {
    for (int j = 0; j < 1; j++) {
        for (int k = 0; k < n; k++) {
            logits[i][j] += X[i][k] * w[k];
        }
    }
}
return logits;
```

3.2.3.3 loss()

```
double LogisticRegression::loss (
                vector< int > y,
                vector< vector< double > > z )
```

Функция подсчитывает ¨функцию потерь¨ нашей логистической регрессии с использованием l2 - регуляризации, основываясь на y исходе(outcome) и полученных сигмоидах.

Аргументы

y	Вектор типа int, содержащий в себе итог(outcome) для нашей выборки
z	Двумерный вектор типа double, содержащий в себе сигмоиды для каждого логита

Возвращает

Число типа double показывающее текущее значение функции потерь

Формула по который мы вычисляем функцию потерь:

$$\sum_{i=0}^{n}(y_i * \log(\sigma_i) + (1 - y_i) * \log(1 - \sigma_i)) + \sum_{i=0}^{l} w_i^2;$$

```
double loss = 0;
for (int i = 0; i < y.size(); i++) {
    loss += (y[i] * (log(z[i][0])) + (1 - y[i]) * log(1 - z[i][0]));
}
for (double i: w_)
    loss += pow(i, 2);

loss /= y.size();
loss *= -1;
return loss;
```

3.2.3.4 model_accuracy()

```
int LogisticRegression::model_accuracy (
                vector< int > results,
                vector< int > y )   [static]
```

Функция показывает точность модели в виде процентов

Аргументы

results	вектор предсказанных результатов тестовой выборки типа int
y	вектор действительных результатов тестовой выборки типа int

Возвращает

целое число типа int, показывающее точность модели

3.2.3.5 predict()

```
vector< int > LogisticRegression::predict (
                const vector< vector< double > > & feauters,
                double threshold = 0.5 )   [static]
```

Функция возвращает предсказания модели по предоставленной выборке

Аргументы

feauters	двумерный вектор типа double, содержащий в себе выборку
threshold	переменная типа double

Возвращает

вектор предсказаний типа int

3.2.3.6 predict_proba()

```
vector< vector< double > > LogisticRegression::predict_proba (
                vector< vector< double > > feauters )   [static]
```

Функция выдает сигмоиды по выборке с использованием наилучших весов, наша функция предсказаний

Аргументы

feauters	двумерный вектор типа double, содержащий в себе выборку

Возвращает

двумерный вектор типа double, содержащий в себе сигмоиды для каждого логита нашей выборки

3.2.3.7 save_weights()

```
void LogisticRegression::save_weights (
            const vector< double > & weights )  [static]
```

Функция сохраняет наилучшие найденные веса в виде weights.txt файла, основываясь на функции потерь

Принцип сохранения весов в функции fit():

```
f((!losses_.empty()) && (loss(y_, z) < *min_element(losses_.begin(), losses_.end()))) {
    save_weights(w_);
}
losses_.push_back(loss(y_, z));
```

Аргументы

weights	двумерный вектор весов типа double

3.2.3.8 saveLossToCSV()

```
void LogisticRegression::saveLossToCSV (
            const vector< double > & losses )  [static]
```

Функция сохраняет ¨функции потерь¨ нашей модели на каждой итерации в файл losses_.csv вида loss,iteration:

1,0

0.80,1

0.5,2

...

n1, n2

Аргументы

losses	веткор потерь нашей модели на каждой итерации типа double

3.2.3.9 sigmoid()

```
vector< vector< double > > LogisticRegression::sigmoid (
```

vector< vector< double > > logits) [static]

Функция подсчитывает сигмоиды по полученным логитам

Аргументы

logits	Двумерный вектор логитов типа double

Возвращает

двумерный вектор типа double, содержащий в себе сигмоиды для каждого логита

Формула по который мы вычисляем логиты:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

```
vector<vector<double» sigmoids;
 for (int i = 0; i < logits.size(); i++) {
     vector<double> sigmoid;
     sigmoid.reserve(logits[0].size());
     for (int j = 0; j < logits[0].size(); j++)
         sigmoid.push_back(1 / (1 + exp(-logits[i][j])));

     sigmoids.push_back(sigmoid);
 }
 return sigmoids;
```

3.2.4 Данные класса

3.2.4.1 losses_

vector<double> LogisticRegression::losses_ [private]

Вектор типа double, содержащий в себе результаты функции потерь на каждой итерации модели при обучении

3.2.4.2 lr_

double LogisticRegression::lr_ = 0 [private]

Переменная типа double, содержащая в себе коэффициент скорости обучения модели

3.2.4.3 max_iter_

int LogisticRegression::max_iter_ = 0 [private]

Переменная типа int, содержащая в себе кол-во итераций, которые будет совершать модель при обучении

3.2.4.4 w_

vector<double> LogisticRegression::w_ [private]

Вектор типа double, содержащий в себе веса наней модели

3.2.4.5 X_

vector<vector<double> > LogisticRegression::X_ [private]

Двумерный вектор типа double, содержащий в себе нормализованную выборку

3.2.4.6 y_

vector<int> LogisticRegression::y_ [private]

Вектор типа int, содержащий в себе итог(outcome)

Объявления и описания членов классов находятся в файлах:

- prj.lab/regression/include/regression/regression.hpp
- prj.lab/regression/regression.cpp

3.3 Класс Plot

Класс Plot предназначен для создания .tex файла с информацией о модели

#include <plot.hpp>

Открытые члены

- Plot ()=default

Открытые статические члены

- static void CreateLatexFile (int max_iter=100, vector< int > results={}, vector< int > y={})

3.3.1 Подробное описание

Класс Plot предназначен для создания .tex файла с информацией о модели

Класс Plot предоставляет возможность вывести данные о текущей модели в виде .tex файла, а именно: функция потерь и confusion matrix.

3.3.2 Конструктор(ы)

3.3.2.1 Plot()

Plot::Plot ( ) [default]

Конструктор по умолчанию для создания объекта класса

3.3.3 Методы

3.3.3.1 CreateLatexFile()

```
void Plot::CreateLatexFile (
            int max_iter = 100,
            vector< int > results = {},
            vector< int > y = {} )   [static]
```

Функция создает statistic.tex файл по результатам работы модели модели, содержащий в себе функцию потерь и confusion matrix

Аргументы

max_iter	переменная типа int, содержащая в себе кол-во итераций, которая совершила модель при обучении (по умолчанию 100)
results	вектор типа int, содержащий в себе результаты предсказаний модели (по умолчанию пустой)
y	вектор типа int, содержащий в себе действительный результат (по умолчанию пустой) Пример:

```
int YESaYES = 0;
int YESaNO = 0;
int NOaNO = 0;
int NOaYES = 0;
for (int i = 0; i < results.size(); i++) {
    if (results[i] == 1 && y[i] == 1) {
        YESaYES++;
    } else if (results[i] == 1 && y[i] == 0) {
        YESaNO++;
    } else if (results[i] == 0 && y[i] == 0) {
        NOaNO++;
    } else if (results[i] == 0 && y[i] == 1) {
        NOaYES++;
    }
}
const string &filename = "statistic.tex";
ofstream outputFile(filename);
if (outputFile.is_open()) {
    outputFile << "\\documentclass{article}\n"
                "\\usepackage{pgfplots}\n"
                "\\usepackage{csvsimple}\n"
                "\\usepackage{pgfplotstable}\n"
                "\\usepackage{array}\n"
                "\\usepackage{graphicx}\n"
                "\\usepackage{multirow}\n"
                "\\usepackage{float} % Add the float package\n"
                "\n"
                "\\newcommand\\MyBox[2]{\n"
                "  \\fbox{\\lower0.75cm\n"
                "    \\vbox to 1.7cm{\\vfil\n"
                "      \\hbox to 1.7cm{\\hfil\\parbox{0.4cm}{#1}\\hfil}\n"
                "      \\vfil}%\n"
                "  }%\n"
                "}\n"
                "\n"
                "\\begin{document}\n"
                "\\title{Model Statistic}\n"
                "\\maketitle\n"
                "\n"
                "\\vspace{1cm}\n"
                "\n"
                "\\begin{figure}[H] % Use the H specifier from the float package\n"
                "  \\centering\n"
                "  \\begin{tikzpicture}\n"
                "    \\begin{axis}[\n"
                "      xlabel={Iters},\n"
                "      ylabel={Log Loss},\n"
                "      xtick=data,\n"
                "      xticklabel={\\pgfmathprintnumber[int detect]{\\tick}},"
                "      xmin=0, xmax=" + to_string(max_iter) + ",\n"
                "      ymin=0, ymax=1,\n"
                "      grid=both,\n"
                "      width=14cm, height=8cm,\n"
                "      samples=100\n"
                "    ]\n"
                "    \n"
                "    \\pgfplotstableread[col sep=comma]{losses_.csv}\\datatable\n"
                "    \\addplot[blue] table[x index=1, y index=0] {\\datatable};\n"
                "    \n"
                "    \\end{axis}\n"
                "  \\end{tikzpicture}\n"
                "  \\caption{Loss changes every iteration}\n"
                "\\end{figure}\n"
                "\n"
                "\\vspace{1cm}\n"
                "\n"
                "\\begin{figure}[H] % Use the H specifier from the float package\n"
                "  \\centering\n"
                "  \\renewcommand\\arraystretch{1.5}\n"
                "  \\setlength\\tabcolsep{0pt}\n"
                "  \\begin{tabular}{c >{\\bfseries}r @{\\hspace{0.7em}}c @{\\hspace{0.4em}}c @{\\hspace{0.7em}}l}\n"
                "    \\multirow{10}{*}{\\rotatebox{90}{\\parbox{1.1cm}{\\bfseries\\centering actual\\\\ value}}} & \n"
                "      & \\multicolumn{2}{c}{\\bfseries Prediction outcome} & \\\\\n"
```

```
                "    & & \\bfseries p & \\bfseries n & \\bfseries total \\\\\\n"
                "    & p$'$ & \\MyBox{" + to_string(YESaYES) +
                "}{Positive} & \\MyBox{" + to_string(NOaYES) +
                "}{Positive} & P$'$ \\\\\[2.4em]\n"
                "    & n$'$ & \\MyBox{" + to_string(YESaNO) + "}{Positive} & \\MyBox{" + to_string(NOaNO) +
                "}{Positive} & N$'$ \\\\\\n"
                "    & total & P & N &\n"
                "  \\end{tabular}\n"
                "  \\caption{Confusion Matrix}\n"
                "\\end{figure}\n"
                "\n"
                "\\end{document}";
} else {
    cerr « "Unable to open file: " « filename « endl;
}
```

Объявления и описания членов классов находятся в файлах:

- prj.lab/plot/include/plot/plot.hpp
- prj.lab/plot/plot.cpp

Глава 4

Файлы

4.1 diabetes_data.hpp

```
00001 //
00002 // Created by hedge on 13.05.2023.
00003 //
00004
00005 #pragma once
00006 #ifndef DIABETES_AI_HPP_09052023DDAT
00007 #define DIABETES_AI_HPP_09052023DDAT
00008
00009 #include <iostream>
00010 #include <utility>
00011 #include <vector>
00012 #include <fstream>
00013 #include <sstream>
00014 #include <string>
00015 #include <cmath>
00016
00017
00018 using namespace std;
00019
00024 class DiabetesData {
00025 public:
00029     DiabetesData() = default;
00036     explicit DiabetesData(const string &file_path, const string &data_name);
00037
00043     explicit DiabetesData(const vector<vector<double» &features);
00049     void load_data_from_file(const string &file_path, const string &data_name);
00054     static vector<vector<double» data_normalization(vector<vector<double» X);
00055
00060     vector<vector<double» get_X ();
00065     vector<int> get_y ();
00066
00067 private:
00071     vector<vector<double» features_;
00075     vector<vector<string» dataset_;
00079     vector<vector<double» X_;
00083     vector<int> y_;
00084 };
00085
00086 #endif
```

4.2 plot.hpp

```
00001 //
00002 // Created by hedge on 09.05.2023.
00003 //
00004 #pragma once
00005 #ifndef DIABETES_AI_HPP_09052023PLOT
00006 #define DIABETES_AI_HPP_09052023PLOT
00007
00008 #include <iostream>
00009 #include <vector>
00010 #include <fstream>
00011 #include <string>
00012
```

```
00013
00014 using namespace std;
00015
00021 class Plot {
00022 public:
00026     Plot() = default;
00038     static void CreateLatexFile(int max_iter = 100, vector<int> results = {}, vector<int> y = {});
00039 private:
00040 };
00041
00042 #endif
```

4.3 regression.hpp

```
00001 //
00002 // Created by hedge on 09.05.2023.
00003 //
00004
00005 #pragma once
00006 #ifndef DIABETES_AI_HPP_09052023REG
00007 #define DIABETES_AI_HPP_09052023REG
00008
00009 #include <iostream>
00010 #include <utility>
00011 #include <vector>
00012 #include <fstream>
00013 #include <sstream>
00014 #include <string>
00015 #include <cmath>
00016
00017 using namespace std;
00018
00023 class LogisticRegression {
00024 public:
00030     LogisticRegression(const vector<vector<double» &X, const vector<int> &y);
00031
00043     static vector<vector<double» logit(vector<vector<double» X, vector<double> w);
00044
00055     static vector<vector<double» sigmoid(vector<vector<double» logits);
00056
00064     vector<double> fit(int max_iter = 100, double lr = 0.1);
00065
00077     double loss(vector<int> y, vector<vector<double» z);
00078
00091     static void save_weights(const vector<double> &weights);
00092
00098     static vector<vector<double» predict_proba(vector<vector<double» feauters);
00099
00106     static vector<int> predict(const vector<vector<double» &feauters, double threshold = 0.5);
00107
00114     static int model_accuracy(vector<int> results, vector<int> y);
00115
00133     static void saveLossToCSV(const vector<double> &losses);
00134
00135 private:
00139     vector<vector<double» X_;
00143     vector<int> y_;
00147     vector<double> w_;
00151     vector<double> losses_;
00155     int max_iter_ = 0;
00161     double lr_ = 0;
00162 };
00163
00164 #endif
```

4.4 Файл prj.test/libtest.cpp

Тестовое приложение библиотеки DiabetesAI.

#include <regression/regression.hpp>
#include <diabetes_data/diabetes_data.hpp>
#include <plot/plot.hpp>

Функции

- void main (int argc, char *argv[])

4.4.1 Подробное описание

Тестовое приложение библиотеки DiabetesAI.

В данном приложении мы тестируем основные функции и методы библиотеки, а именно: обработка датасета с помощью класса DiabetesData, обучение модели с помощью класса LogisticRegression по встроенному датасету 1 и кол-вом итераций = 10, вывод функции потерь в консоль, прогнозирование диабета по встроенному датасету 2 и вывод статистики модели с помощью класса Plot

4.4.2 Функции

4.4.2.1 main()

```
void main (
                int argc,
                char * argv[] )
```

Аргументы

argc	Название .exe файла для запуска тестового приложения
argv	полный путь до папки с датасетами вида ¨C:\...\data¨

Обрабатываем встроенный датасет 1
```
DiabetesData a(argv[1],"dataset1");
vector<vector<double» X1 = a.get_X();
vector<int> y1 = a.get_y();
LogisticRegression lg1(X1, y1);
```

Выводим функции потерь в консоль
```
vector<double> losses = lg1.fit(10);
for (double losse: losses)
    cout « losse « endl;
```

Обрабатываем встроенный датасет 2
```
DiabetesData b(argv[1],"dataset2");
vector<vector<double» X = a.get_X();
vector<int> y = a.get_y();
```

Берем предсказания датасета 2 из обученной модели
```
LogisticRegression lg2(X, y);
vector<int> results = LogisticRegression::predict(X);
```

Создаем statistic.tex файл
```
Plot::CreateLatexFile(results, y, 10);
```

Предметный указатель