20220424-书&C++

1.过程描述

1.1 Complications一书

> Doctors belong to an insular world—one of hemorrhages and lab tests and people sliced open. We are for the moment the healthy few who live among the sick. And it is easy to become alien to the experiences and sometimes the values of the rest of civilization.

或许应该在某种程度上体谅一些医生态度的恶劣

> We are all, whatever we do, in the hands of flawed human beings. The fact is hard to stare in the face. But it is inescapable. Every doctor has things he or she ought to know but has yet to learn, capacities of judgment that will fail, a strength of character that can break.

医生也是普通人

1.2 C++ on machine learning

data.h C++ 复制代码

```cpp
#pragma once
#ifndef _DATA_H
#define _DATA_H

#include <vector>
#include <stdint.h>
#include <stdlib.h>

class data
{
    std::vector<uint8_t>* feature_vector;
    uint8_t label;
    int enum_label;
public:
    data();
    ~data();
    void set_feature_vector(std::vector<uint8_t>*);
    void append_to_feature_vector(uint8_t);
    void set_label(uint8_t);
    void set_enumerated_label(int);

    int get_feature_vector_size();
    uint8_t get_label();
    uint8_t get_enumerated_label();

    std::vector<uint8_t>* get_feature_vector();
};


#endif // ! _DATA_H
```

data.cpp C++ 复制代码

```cpp
#include "data.h"

data::data()
{
    feature_vector = new std::vector<uint8_t>;
}

data::~data()
{
}

void data::set_feature_vector(std::vector<uint8_t>* vect)
{
    feature_vector = vect;
}

void data::append_to_feature_vector(uint8_t val)
{
    feature_vector->push_back(val);
}

void data::set_label(uint8_t val)
{
    label = val;
}

void data::set_enumerated_label(int val)
{
    enum_label = val;
}

int data::get_feature_vector_size()
{
    return feature_vector->size();
}

uint8_t data::get_label()
{
    return label;
}

uint8_t data::get_enumerated_label()
{
    return enum_label;
}
```

```
46
47  std::vector<uint8_t>* data::get_feature_vector()
48  {
49      return feature_vector;
50  }
51
```

data_handler.h C++ 复制代码

```cpp
#pragma once
#ifndef _DATA_HANDLER_H
#define _DATA_HANDLER_H

#include <fstream>
#include <stdint.h>
#include <vector>
#include <string>
#include <map>
#include <unordered_set>
#include "data.h"


class data_handler
{
    std::vector<data*>* data_array;
    std::vector<data*>* training_data;
    std::vector<data*>* test_data;
    std::vector<data*>* validation_data;

    int num_classes;
    int feature_vector_size;
    std::map<uint8_t, int> class_map;

    const double TRAIN_SET_PERCENT = 0.75;
    const double TEST_SET_PERCENT = 0.20;
    const double VALIDATION_SET_PERCENT = 0.05;

public:
    data_handler();
    ~data_handler();

    void read_feature_vector(std::string path);
    void read_feature_labels(std::string path);
    void split_data();
    void count_classes();

    uint32_t convert_to_little_endian(const unsigned char* bytes);

    std::vector<data*>* get_training_data();
    std::vector<data*>* get_test_data();
    std::vector<data*>* get_validation_data();
};

#endif
```

data_handler.cpp C++ 复制代码

```cpp
#include "data_handler.h"
#pragma warning(disable:4996)
#include <iostream>
data_handler::data_handler()
{
    data_array = new std::vector<data*>;
    test_data = new std::vector<data*>;
    training_data = new std::vector<data*>;
    validation_data = new std::vector<data*>;
}

data_handler::~data_handler()
{
}

void data_handler::read_feature_vector(std::string path)
{
    uint32_t header[4];
    unsigned  char bytes[4];
    FILE* f = fopen(path.c_str(), "r");
    if (f)
    {
        for(int i = 0; i < 4; i++)
        {
            if (fread(bytes, sizeof(bytes), 1, f))
            {
                header[i] = convert_to_little_endian(bytes);
            }
        }
        printf("Done getting file header.\n");
        int image_size = header[2] * header[3];
        for (int i = 0; i < header[1]; i++)
        {
            data* d = new data();
            uint8_t element[1];
            //uint8为一个字节
            for (int j = 0; j < image_size;j++)
            {
                if (fread(element, sizeof(element), 1, f))
                {
                    std::cout << j<<std::endl;
                    d->append_to_feature_vector(element[0]);
                }
                else
                {
```

```
                    printf("Error reading from file.\n");
                    exit(1);
                }
            }
            data_array->push_back(d);
        }
        printf("successfully read and store feature vectors.\n", data_array->size());
    }
    else
    {
        printf("could not find file.\n");
        exit(1);
    }
}

void data_handler::read_feature_labels(std::string path)
{
    uint32_t header[2];
    unsigned  char bytes[4];
    FILE* f = fopen(path.c_str(), "r");
    if (f)
    {
        for (int i = 0; i < 2; i++)
        {
            if (fread(bytes, sizeof(bytes), 1, f))
            {
                header[i] = convert_to_little_endian(bytes);
            }
        }
        printf("Done getting label file header.\n");
        for (int i = 0; i < header[1]; i++)
        {
            uint8_t element[1];

            if (fread(element, sizeof(element), 1, f))
            {
                data_array->at(i)->set_label(element[0]);
            }
            else
            {
                printf("Error reading from file.\n");
                exit(1);
            }
        }
        printf("successfully read and store label.\n");
    }
    else
```

```
93      {
94          printf("could not find file.\n");
95          exit(1);
96      }
97  }
98
99  void data_handler::split_data()
100 {
101     std::unordered_set<int> used_indexes;
102     int train_size = data_array->size() * TRAIN_SET_PERCENT;
103     int test_size = data_array->size() * TEST_SET_PERCENT;
104     int valid_size = data_array->size() * VALIDATION_SET_PERCENT;
105
106     //Training data
107     int count = 0;
108     while (count < train_size)
109     {
110         int random_index = rand() % data_array->size();
111         if (used_indexes.find(random_index) == used_indexes.end())
112         {
113             training_data->push_back(data_array->at(random_index));
114             used_indexes.insert(random_index);
115             count++;
116         }
117     }
118     //Test data
119     count = 0;
120     while (count < test_size)
121     {
122         int random_index = rand() % data_array->size();
123         if (used_indexes.find(random_index) == used_indexes.end())
124         {
125             test_data->push_back(data_array->at(random_index));
126             used_indexes.insert(random_index);
127             count++;
128         }
129     }
130     //Validation data
131     count = 0;
132     while (count < valid_size)
133     {
134         int random_index = rand() % data_array->size();
135         if (used_indexes.find(random_index) == used_indexes.end())
136         {
137             validation_data->push_back(data_array->at(random_index));
138             used_indexes.insert(random_index);
139             count++;
140         }
```

```
141         }
142         printf("Training data size: %lu.\n", training_data->size());
143         printf("Test data size: %lu.\n", test_data->size());
144         printf("Validation data size: %lu.\n", validation_data->size());
145     }
146
147     void data_handler::count_classes()
148     {
149         int count = 0;
150         for (unsigned i = 0; i < data_array->size(); i++)
151         {
152             if (class_map.find(data_array->at(i)->get_label()) ==
    class_map.end())
153             {
154                 class_map[data_array->at(i)->get_label()] = count;
155                 data_array->at(i)->set_enumerated_label(count);
156                 count++;
157             }
158         }
159         num_classes = count;
160         printf("successfully extracted %d unique classes.\n", num_classes);
161     }
162
163     uint32_t data_handler::convert_to_little_endian(const unsigned char*
    bytes)
164     {
165         return (uint32_t)((bytes[0] << 24) |
166             (bytes[1] << 16) |
167             (bytes[2] << 8) |
168             (bytes[3]));
169     }
170
171     std::vector<data*>* data_handler::get_training_data()
172     {
173         return training_data;
174     }
175
176     std::vector<data*>* data_handler::get_test_data()
177     {
178         return test_data;
179     }
180
181     std::vector<data*>* data_handler::get_validation_data()
182     {
183         return validation_data;
184     }
185
186     int main()
```

```
{
    data_handler* dh = new data_handler();
    //dh->read_feature_vector("train-images.idx3-ubyte");
    dh->read_feature_labels("train-labels.idx1-ubyte");
    dh->split_data();
    dh->count_classes();
}
```

2.结果输出

昨天忘了写结果了，这两天的活动基本类似，上午基本在看书，下午在copy spdlog的代码，希望能通过这个过程学到一些C++的编程范式。晚上开始看一个新近发现的C++ ML视频，由于是从头开始的，感觉应该能比较有收获。但不知道为啥代码跑不起来，似乎是在读取图片数据的时候出了问题。明天再看下。