

NekoBytes Week8 I/O & AI Introduction

Basic Ideas of Programming Fall 2024

cs-wiki

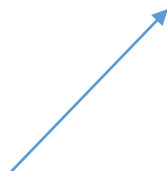
Part1.We need Save C标准I/O库介绍



这把贪吃蛇玩了半天3k
分了，存个档下次继续！

我没写存档功能

情景导入



看起来很眼熟？
这种行为在steam上一直在发生！
想让你的贪吃蛇也具有一样的读取模组/存档的能力？
那么除非你作出人生中最重要的一抉择之一——学习C语言I/O库！

Agenda

- **C语言I/O库概述**
- 标准输入输出函数
- 文件操作
- 标准输入输出和错误输出
- 输入输出重定向

C语言I/O库概述



什么是<stdio.h>?

stdio.h是C标准库中的一个头文件，提供了输入输出功能的定义和声明。它是“标准输入输出头文件”（Standard Input Output Header）的缩写。stdio.h中包含了许多函数和宏，这些函数和宏用于处理文件、控制台输入输出、格式化输入输出等操作

Agenda

- C语言I/O库概述
- **标准输入输出函数**
- 文件操作
- 标准输入输出和错误输出
- 输入输出重定向

标准输入输出函数

函数名: printf()

原型: int printf(const char *format, ...);

```
C test.c > main()
1 #include<stdio.h>
2 int main()
3 {
4     int a = 114514;
5     printf(_Format: "hello world %d",a);
6     return 0;
7 }
```

问题 输出 调试控制台 终端 端口

code cd "c:\Users\31669\Desktop\code\" ; if (\$?) { gcc t
hello world 114514

函数名: scanf()

原型: int scanf(const char *format, ...);

```
C test.c > ...
1 #include<stdio.h>
2 int main()
3 {
4     int a;
5     scanf(_Format: "%d",&a);
6     printf(_Format: "读入了变量a,他的值为:%d",a);
7     return 0;
8 }
9
```

问题 输出 调试控制台 终端 端口

code cd "c:\Users\31669\Desktop\code\" ; if (\$?) { gcc t
114514
读入了变量a,他的值为:114514

标准输入输出函数

函数名: getchar ()
原型: int getchar(void);

```
C test.c > ...
1  #include<stdio.h>
2  int main()
3  {
4      char a;
5      a=getchar();
6      printf(_Format: "读入了变量a,他的值为:%c",a);
7      return 0;
8  }
9
```

问题 输出 调试控制台 终端 端口

- code cd "c:\Users\31669\Desktop\code\" ; if (\$?) { gcc Kitty! You can have a cheeze burger!
读入了变量a,他的值为:k

函数名: gets() (请少用这玩意)
原型: char *gets(char *str);

```
C test.c > ...
1  #include<stdio.h>
2  int main()
3  {
4      char a[100];
5      gets(a);
6      printf(_Format: "读入了变量a,他的值为:%s",a);
7      return 0;
8  }
9
```

问题 输出 调试控制台 终端 端口

- code cd "c:\Users\31669\Desktop\code\" ; if (\$?) { gcc NekoBytes!
读入了变量a,他的值为:NekoBytes!

标准输入输出函数

函数名: puts ()
原型: int puts(const char *str);

```
C test.c > ...
1  #include<stdio.h>
2  int main()
3  {
4      puts(_Buffer: "欢迎加入计科协");
5      return 0;
6  }
```

问题 输出 调试控制台 终端 端口

code cd "c:\Users\31669\Desktop\code\" ; if
欢迎加入计科协

函数名: putchar()
原型: int putchar(int char);

```
C test.c > main()
1  #include<stdio.h>
2  int main()
3  {
4      putchar(_Character: 'G');
5      return 0;
6  }
```

问题 输出 调试控制台 终端 端口

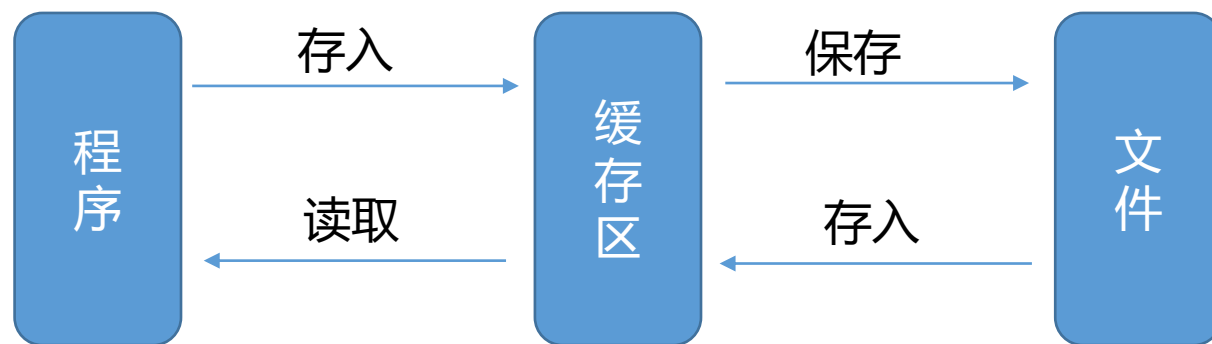
code cd "c:\Users\31669\Desktop\code\"
G

Agenda

- C语言I/O库概述
- 标准输入输出函数
- **文件操作**
- 标准输入输出和错误输出
- 输入输出重定向

文件操作

缓存 (buffer)



Why 缓存?

我什么都不知道



文件操作

操作句柄 FILE*

```
#include<stdio.h>
int main()
{
    FILE* fp;
    //只读打开data1
    fp = fopen(_FileName: "data1.txt",_Mode: "r");
    //关闭释放缓存
    fclose(_Stream: fp);
}
```

fopen 函数参数 mode 总结:

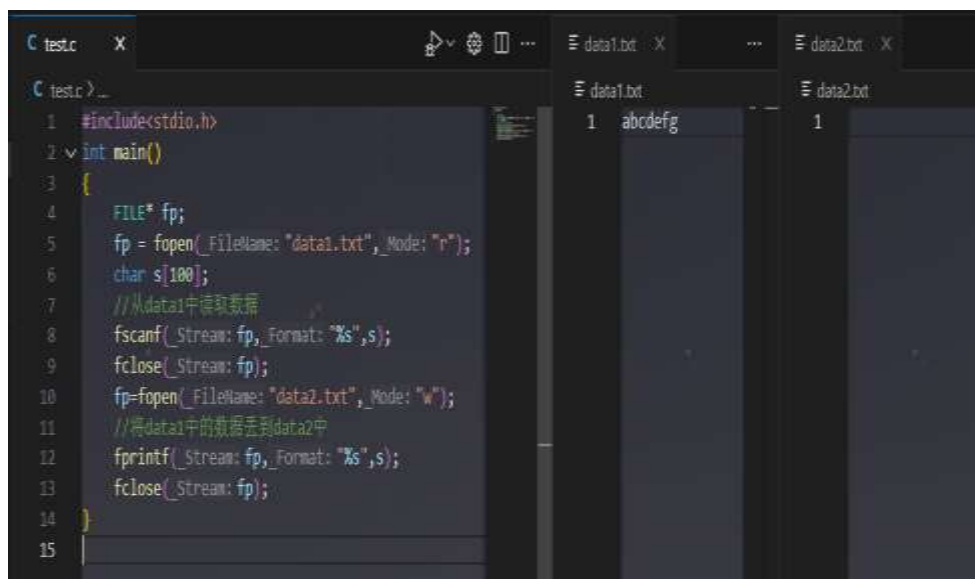
- "r": 只读, 文件必须存在。
- "w": 只写, 如果不存在则创建, 存在则覆盖。
- "a": 追加, 如果不存在则创建。
- "r+": 允许读和写, 文件必须存在。
- "w+": 允许读和写, 文件不存在则创建, 存在则覆盖。
- "a+": 允许读和追加, 文件不存在则创建。

文件操作

fpinrt() & fscanf()

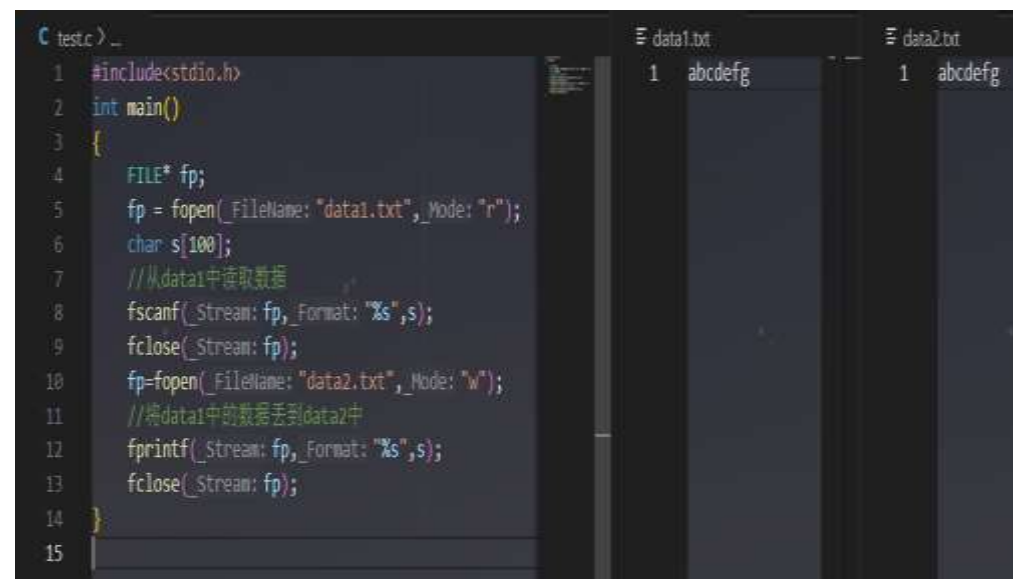
```
int __cdecl fprintf(FILE *const _Stream, const char *const _Format, ...)
```

```
int __cdecl fscanf(FILE *const _Stream, const char *const _Format, ...)
```



```
C test.c X
C test.c > ...
1 #include<stdio.h>
2 int main()
3 {
4     FILE* fp;
5     fp = fopen(_FileName: "data1.txt", _Mode: "r");
6     char s[100];
7     //从data1中读取数据
8     fscanf(_Stream: fp, _Format: "%s", s);
9     fclose(_Stream: fp);
10    fp=fopen(_FileName: "data2.txt", _Mode: "w");
11    //将data1中的数据丢到data2中
12    fprintf(_Stream: fp, _Format: "%s", s);
13    fclose(_Stream: fp);
14 }
15
```

The screenshot shows a C program in a code editor. The program opens 'data1.txt' in read mode, reads a string 's' using `fscanf`, closes the file, opens 'data2.txt' in write mode, and writes the string 's' using `fprintf`. The file explorer on the right shows 'data1.txt' containing '1 abcdefg' and 'data2.txt' containing '1'.



```
C test.c > ...
1 #include<stdio.h>
2 int main()
3 {
4     FILE* fp;
5     fp = fopen(_FileName: "data1.txt", _Mode: "r");
6     char s[100];
7     //从data1中读取数据
8     fscanf(_Stream: fp, _Format: "%s", s);
9     fclose(_Stream: fp);
10    fp=fopen(_FileName: "data2.txt", _Mode: "w");
11    //将data1中的数据丢到data2中
12    fprintf(_Stream: fp, _Format: "%s", s);
13    fclose(_Stream: fp);
14 }
15
```

The screenshot shows a C program in a code editor. The program opens 'data1.txt' in read mode, reads a string 's' using `fscanf`, closes the file, opens 'data2.txt' in write mode, and writes the string 's' using `fprintf`. The file explorer on the right shows 'data1.txt' containing '1 abcdefg' and 'data2.txt' containing '1 abcdefg'.

当然还有 fread()&fwrite()

```
size_t __cdecl fread(void *_Buffer, size_t _ElementSize, size_t _ElementCount, FILE *_Stream)
```

```
size_t __cdecl fwrite(const void *_Buffer, size_t _ElementSize, size_t _ElementCount, FILE *_Stream)
```

Agenda

- C语言I/O库概述
- 标准输入输出函数
- 文件操作
- **标准输入输出和错误输出**
- 输入输出重定向

标准输入输出和错误输出

- 标准输入：
 stdin
- 标准输出：
 stdout
- 标准错误输出：
 stderr

在用户程序启动时，main 函数还没开始执行之前，会自动打开三个 FILE* 指针分别是：stdin、stdout、stderr，这三个文件指针是 libc 中定义的全局变量，在 stdio.h 中声明，printf 向 stdout 写，而 scanf 从 stdin 读，用户程序也可以直接使用这三个文件指针。

Agenda

- C语言I/O库概述
- 标准输入输出函数
- 文件操作
- 标准输入输出和错误输出
- **输入输出重定向**

输入输出重定向

你是否为大量输入数据复制不完而苦恼？

是否为大量输出数据超过终端容纳范围而悲伤？

是否为文件读写语法复杂而眉头紧锁？

没关系，没关系，只要一个小小的“<”或者“>”符号，就能够在终端完成输入输出的重定向！解决你的烦恼！



fscanf & fprintf



fread & fwrite



fgets & fputs



```
./a.exe > data1.txt
```


输出重定向

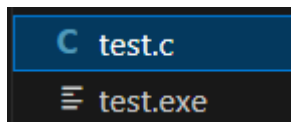
测试程序（当然实际要输出的东西可以非常多）

```
#include<stdio.h>
int main()
{
    printf(_Format: "12345");
}
```

使用**Ctrl+`** 打开vscode终端，随后用gcc将.c文件编译为.exe文件

```
code gcc ./test.c -o test.exe
```

```
@KX:~/coding$ gcc test.c -o test.out
```

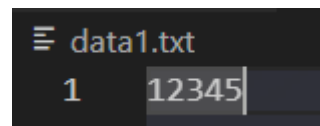


想想哪个是Win哪个是Linux?

优雅地用终端完成输出重定向

```
code ./test.exe > data1.txt
@KX:~/coding$ ./test.out > data.txt
```

然后我们地结果就理所当然地输出到了data1.txt



输入重定向

测试文件和测试程序

```
test.c >...
1 #include <stdio.h>
2 int main()
3 {
4     int a;
5     scanf(_Format: "%d",&a);
6     printf(_Format: "%d",a);
7 }
```

```
data1.txt
1 12345
```

使用**Ctrl+`** 打开vscode终端，随后用gcc将.c文件编译为.exe文件

```
code gcc ./test.c -o test.exe
code gcc ./test.c -o test.out
```

```
C test.c
test.exe
```

然后，我们使用与输出重定向类似地方式重定向输入，然后非常的amazing，G了！

```
31669 code ./test.exe < data1.txt
ParserError:
Line |
1    | ./test.exe < data1.txt
      | ~
      | The '<' operator is reserved for future use.
```

查阅资料后发现，上面的方法在最新的powershell中已经不支持了，但是在Linux/Unix系统仍然适用，在windows下，我们需要：

```
~/coding$ ./test.out < data.txt
12345
```

← linux

```
code Get-Content data1.txt | ./test.exe
12345
```

同时重定向输入输出

我们保留之前重定向输入的测试程序，但是修改终端的代码：

```
code Get-Content data1.txt | ./test.exe >> data1.txt
```

```
@KX:~/coding$ ./test.out < data.txt >> data.txt
```

其中 “>>” 代表在data1.txt文件后继续添加要输入的内容

```
data.txt
```

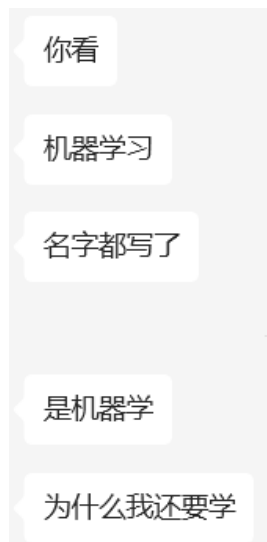
```
1 1234512345
```



Cat Break! ! !



NekoBytes Week8 I/O & AI Introduction



Part2.Why I still need Study 人工智能&机器学习导论

Agenda

- **AI vs ML**
- What can AI help me
- 机器学习的主要类型
- 线性回归的基本讲解

AI vs ML

	AI(人工智能 Artificial Intelligence)	ML(机器学习 Machine learning)
定义	模拟人类智能进行任务的能力	通过数据和算法让系统自动学习和改进
方法	包括专家系统、逻辑推理、遗传算法等	主要使用统计学习方法，如线性回归、神经网络等
应用	广泛应用于多种智能任务和系统	主要用于数据分析、预测和模式识别
趋势	多模态AI、通用人工智能（AGI）等	自监督学习、迁移学习、边缘计算等

Agenda

- AI vs ML
- **What can AI help me**
- 机器学习的主要类型
- 线性回归的基本讲解

What can AI help me

- 代码debug
- 快速给出设计思路
- 翻译看不懂的外文文献
- 快速总结冗长的各类报告
- 提供类搜索引擎服务而减少筛选答案的时间
- 讲解自己做不出的大雾（或者其他科）题目
- 在完全不听的情况下完成水课作业



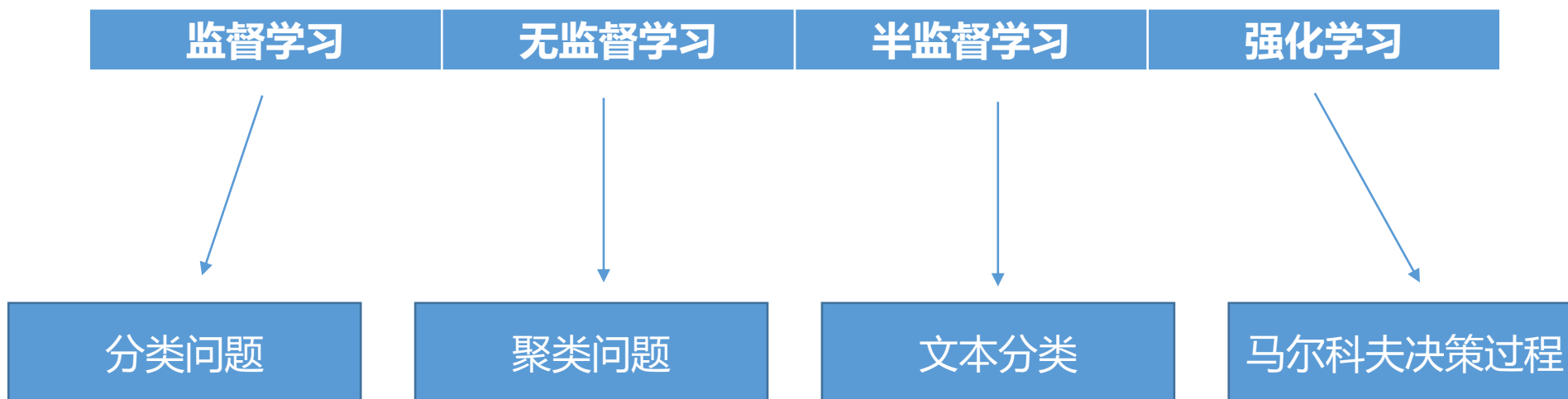
AI搜索引擎与传统的基于关键词索引和链接排名的搜索引擎有所不同。它通过理解用户的查询意图和上下文，直接提供整理好的答案或信息摘要¹。这种方式不仅加速了信息检索的过程，还大幅度提升了搜索结果的准确性和相关性。尽管AI在生成式AI产品方面取得了突破，但谷歌等搜索引擎拥有先发优势和强大的搜索能力，使其在搜索市场上的地位仍然稳固²。总之，AI搜索引擎在改善搜索体验方面具有潜力，但传统搜索引擎仍然占据主导地位。³



Agenda

- AI vs ML
- What can AI help me
- **机器学习的主要类型**
- 线性回归的基本讲解

机器学习的主要类型



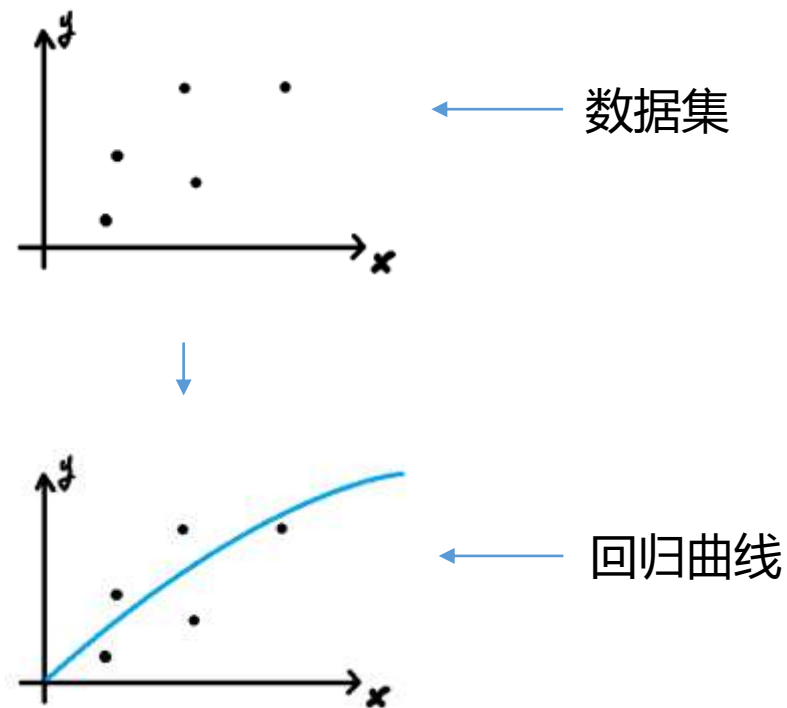
Agenda

- AI vs ML
- What can AI help me
- 机器学习的主要类型
- **线性回归的基本讲解**

线性回归的基本讲解

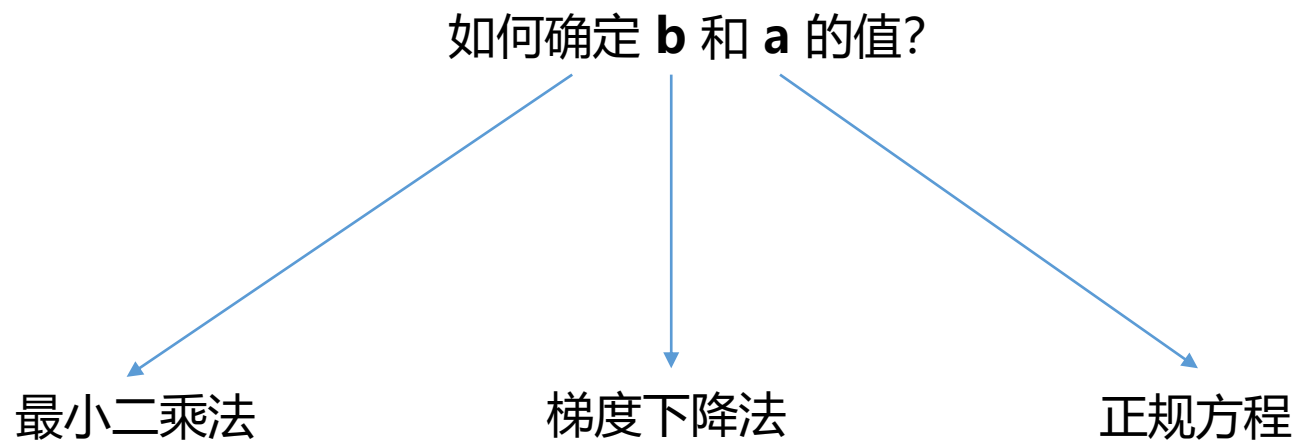
什么是回归分析？

回归分析是指一种预测性的建模技术，主要是研究自变量和因变量的关系。通常使用线/曲线来拟合数据点，然后研究如何使曲线到数据点的距离差异最小。



线性回归的基本讲解

以一元线性回归（即拟合曲线为 $y=bx+a$ ）为例



Ask Time! ! ! !



Homework

1. 读取数据集的线性回归预测方法 (Lab)
2. SoftMax分类或简易神经网络搭建 (Proj3)