一、AI 应用

1、AI 应用开发的 2 个主要部分：

- 机器学习模型开发
- AI 应用开发

核心优势：算法、模型

2、机器学习模型开发

（1）机器学习流程：

- 数据采集和标注
- 数据清洗
- 特征选择
- 模型选择
- 模型训练和测试
- 模型性能评估和优化
- 模型使用

（2）机器学习方法：

- 传统机器学习
  - 俗称机器学习，其中的神经网络逐渐发展到深度学习方法
  - 使用 sklearn 框架
- 深度学习
  - 从传统机器学习的神经网络中发展而来
  - 使用 tensorflow2 框架
  - 擅长领域：计算机视觉（CV）、自然语言处理（NLP）

3、AI 应用开发

- CS 模式（Client/Server）
  - QQ、微信、Foxmail、手机端的 APP 等
- BS 模式（Browser/Server，WEB 开发主流模式）
  - WEB 邮箱、各种网站。本课程主要讲的是 WEB 开发即 BS 模式开发
- 混合模式（APP 开发主流模式）
  - 微信+小程序，目前 APP 中很多是采用 WEB 访问的方式构建功能

4、CS 与 BS 优缺点

- CS 优点：本地功能强大，事务在本地及时处理，界面渲染效果好等
- CS 缺点：本地必须安装客户端软件，而且客户端需要维护升级
- BS 优点：本地不需要安装特定客户端软件，只要有浏览器即可，本地不需要维护升级
- BS 缺点：本地处理业务的能力弱，依赖服务器能力，界面渲染效果不如客户端灵活丰富

二、Flask WEB 应用开发

1、URL 的组成

URL=传输协议+主机名+端口号+（目录）文件名

（1）传输协议一般是 http（超文本传输协议）或 https（安全套接字层超文本传输协议）

超文本传输协议：HyperText Transfer Protocol

安全套接字层超文本传输协议：HyperText Transfer Protocol over Secure Socket Layer

2、Flask 默认自带了一个轻量级的 WEB Server，无需用户单独安装

3、URL 参数传递

- Flask 中如果要传递一个变量或者一个参数，可以通过表单和地址栏两种方式来传递。

其中，通过浏览器地址栏 URL 方式传递/获取某个变量或参数使用得比较多。这样可以使用相同的 URL 指定不同的参数来访问不同的内容

• Flask 通过 URL 传递参数，传递参数的语法是：'/<参数名>/'。

需要注意两点：参数需要放在一对<>（尖括号）内；视图函数中需要设置同 URL 中相同的参数名

示例代码：

```
@app.route("/user/<name>")
def visitByName(name):
  return "接收到的名称为：%s" % name
```

4、URL 反转

有时候在作网页重定向或是模板文件时需要使用在视图函数中定义的 URL，我们必须根据视图函数名称得到当前所指向的 URL，这就是 URL 反转。

使用 URL 反转，用到了 url_for()函数，需要使用 from flask import url_for 导入，url_for()函数最简单的用法是以视图函数名作为参数，返回对应的 URL

5、页面跳转和重定向

Flask 中提供了重定向函数 redirect()，该函数的功能就是跳转到指定的 URL。重定向是将原本的 URL 重新定向成为一个新的 URL，可以实现页面的跳转

6、Jinja2 模板引擎

在 Flask 中通常使用 Jinja 2 模板引擎来实现复杂的页面渲染。

后端实际上实现的功能一般叫做业务逻辑，前端完成的功能一般叫做表现逻辑。如果把业务逻辑和表现逻辑混在一起，势必造成系统耦合度高、代码维护困难的现象，因此分离业务逻辑和表现逻辑，把表现逻辑交给视图引擎，即网页模板

• 模板实质上是一个静态的包含 HTML 语法的全部或片段的文本文件，也可包含由变量表示的动态部分。使用真实值替换网页模板中的变量，生成对应数据的 HTML 片段，这一过程称为渲染。Flask 提供了 Jinja 2 模板引擎来渲染模板，模板文件保存在 templates 目录中

7、渲染模板

Flask 通过 render_template()函数来实现模板的渲染。要使用 Jinja2 模板引擎，需要使用 from flask import render_template 命令导入 render_template 函数。在视图函数的 return 方法中，render_template()函数的首个参数声明使用哪一个模板文件。

（1）if 语句

在 Jinja 2 模板引擎中可以使用 if 和 for 循环控制语句，控制模板渲染的方向。模板引擎中，if 和 for 语句中应该放到{%%}中

基本语法如下：

```
{% if condition1 %} <!– condition1 指的是条件 1-->
{% elif conditio2 %} <!– condition2 指的是条件 2-->
{% else %} <!-- 条件不满足时-->
{% endif %} <!-- 结束 if 语句-->
```

（2）for 语句

模板中的 for 语句定义如下：

```
{% for 目标 in 对象 %}
<p>目标</p>
{% endfor %}
```

注意：不可以使用 continue 和 break 表达式来控制循环的执行。

8、过滤器

过滤器本质上是一个转换函数，有时候不仅需要输出变量的值，还需要把某个变量的值修改后再显示

出来，而在模板中不能直接调用 Python 中的某些方法，这时就用到了过滤器

- 与字符串操作相关的过滤器
- 对列表进行操作相关的过滤器
- 对数值进行操作相关的过滤器

（1）与字符串操作相关的过滤器

- {{name|default('None',true)}

  name 为变量名，如果 name 为空，则用 None 替换 name
- {{'hello'|capitalize}}　　将字符串 hello 转化成 Hello，实现首字母大写的目的
- {{'HELLO'|lowere}}　　将字符 HELLO 全部转为小写
- {{'hello'|replace('h','x')}}　　将 hello 中的字母 h 替换成 x

（2）对列表进行操作相关的过滤器

- {{[01,80,42,44,77]|first}}　　取得列表中的首个元素 01
- {{[01,80,42,44,77]|last}}　　取得列表中的最后一个元素 77
- {{[01,80,42,44,77]|count}}

  取得列表中的元素个素，个数为 5，count 也可以使用 length 替换
- {{[01,80,42,44,77]|sort}}　　列表中的元素重新排序，默认按照升序进行排序
- {{[01,80,42,44,77]|join(',')}}　将列表中的元素合并为字符串，返回 1,80,42,44,77

（3）对数值进行操作相关的过滤器

- {{18.8888|round}}　　四舍五入取得整数，返回 19.0
- {{18.8888|round(2,'floor')}}　　保留小数点后 2 位，返回结果为 18.88
- {{-2|abs}}　　求绝对值运算，返回结果为 2

8.1、宏的使用

（1）• 宏的定义

（2）• 宏的导入

一个宏可以被不同的模板使用，建议将其声明在一个单独的模板文件中。需要使用时导入即可，而导入的方法类似于 Python 中的 import

（3）• include 的使用

（1）• 宏的定义

- jinja 2中的宏功能类似于传统程序语言中的函数，它跟Python中的函数类似，可以传递参数，但是不能有返回值，可以将一些经常用到的代码片段放到宏中，然后把一些不固定的值抽取出来作为一个变量
- 宏(Macro)有声明和调用两个部分:

```
<!--定义宏-->
{% macro input(name, type='text', value= ' ') -%}
        <input type="{{ type }}" name="{{ name }}" value="{{ value|e }}">
{%- endmacro %}
```

上面的代码定义了一个宏，定义宏要加macro，宏定义结束要加endmacro标志。上面宏定义的名称是input，它有3个参数，分别是name、type和value， value参数有默认值。

```
<!--调用宏-->
{{ input('username')}}
{{ input('password',type='password')}}
```

使用：举例

```
{#宏的定义#}
{% macro input(name, type='text', value= '') -%}
    <input type="{{ type }}" name="{{ name }}" value="{{ value|e }}">
{%- endmacro %}

{#宏的使用#}
<p> 用户名 ： {{ input('username')}}</p>
<p> 密 码 ： {{ input('password',type='password')}}</p>
<p> 登 录 ： {{ input('submit',type='submit',value='登录')}}</p>
```

执行网页后，生成对应的代码如下：

```
<p> 用户名 ： <input type="text" name="username" value="">  </p>
<p> 密 码 ： <input type="password" name="password" value="">  </p>
<p> 登 录 ： <input type="submit" name="submit" value="登录">  </p>
```

8.2、set 和 with 语句

- set与with语句都可以在Jinja 2中定义变量并赋予值。set定义的变量在整个模板范围内都有效，with关键字在定义变量并赋值的同时，限制了with定义变量的作用范围。
- set关键字的使用方法：

(1) 给变量赋值：{% set telephone ='1388888888' %}

(2) 给列表或数组赋值：{% set nav = [('index.html', 'index'), ('product.html', 'product)] %}

可以在模板中使用{{telephone}}和{{nav}}来引用这些定义的变量。

- with关键字的使用方法：

```
{% with pass = 60 %}
{{ pass }}
{% endwith %}
```

with定义的变量的作用范围在{%with%}和{%endwith%}代码块内，在模板的其他地方，引用此变量值无效。

9、静态文件加载

静态文件的加载一般需要先新建文件夹 static，在文件夹下再新建 css、js 和 images 文件夹，在这些文件夹中存放 css、js、images 文件，同时要需要使用‘url_for’函数。

10、视图高级技术：路由

路由的概念：在 Flask 应用中，路由是指用户请求的 URL 与视图函数之间的映射，处理 URL 和函数之间关系的程序称为路由。Flask 框架根据 HTTP 请求的 URL 在路由表中匹配预定义的 URL 规则，找到对应的视图函数，并将视图函数的执行结果返回给服务器

11、路由视图函数的绑定

在 Flask 框架中，默认是使用@app.route 装饰器将视图函数和 URL 绑定，例如：

```
@app.route('/')
def hello_world():
    retrun ‘hello world’
```

除了使用@app.route 装饰器，我们还可以使用 add_url_rule 来绑定视图函数和 URL

12、数据交互

- 表单处理

表单是搜集用户数据信息的各种表单元素的集合区域，它的作用是实现用户和服务器的数据交互。通过表单搜集客户端输入的数据信息，然后提交到网站服务器端进行处理。

- 文件上传

  Flask 文件上传有 3 点要求：

- 一个<form>标签被标记有enctype=multipart/form-data，

  并且在里面包含一个<input type=file>标签

- 服务端应用通过请求对象上的files 字典访问文件

- 使用文件的 save()方法将文件永久地保存在文件系统上的某处

注意：表单中必须要有enctype="multipart/form-data"，否则上传文件无效。

一般可以写成：<form action="" method="post" enctype="multipart/form-data">

三、数据处理与分析工具

1、数据处理与分析

- 数据处理
- 数据可视化

2、数据处理工具

- Numpy

Numpy 是 Python 科学计算的基础库，主要提供了高性能的N 维数组实现以及计算能力

- 创建数组

（1）----通过 python 的 list 列表创建数组

```
a = np.array([1 , 2 , 3 , 4])
b = np.array([[1 , 2], [3 , 4], [5 , 6]])
# 查看 array 的属性，包括数据的维度和类型
print(b.ndim) //属性
print(b.shape) //维度
print(b.dtype)
```

（2）----通过 Numpy 的函数创建数组

- 查看数组维度和类型
- 数组元素的访问及索引----->指索引切片操作
- Numpy 的运算

----数组和标量的运算

```
a = np.arange(6)
print(a)
print(a + 5)          # 数组和标量加法
# 创建1~5之间的20个随机整数，形成一维向量，即1×20的数组矩阵
b = np.random.randint(1 , 5 , 20)
print(b)
# 将1×20的数组矩阵构建出4×5的数组矩阵
c = b.reshape(4 , 5)
print(c)
print(c * 3)          # 数组和标量乘法
```

----数组与数组之间的运算

- 数组和数组的运算，如果数组的维度相同，那么在组里对应位置进行逐个元素的数学运算。

```
# 创建5×4数组矩阵，元素是1~5随机整数
a = np.random.random_integers(1 , 5 , (5 , 4 ))
print(a)
# 创建一个5×4的数组矩阵，每个元素都是1
b = np.ones((5 , 4) , dtype=int)
print(b)
# 数组加法
a + b
print(a + b)
```

```
# 创建3×4数组矩阵，元素是1~5的随机整数
c = np.random.random_integers(1 , 5 , ( 3 , 4 ))
print(c)
# 创建3×4的数组矩阵，元素是1~5随机整数
d = np.random.random_integers(1 , 5 , ( 3 , 4 ))
print(d)
# 数组相乘，逐元素相乘， 不是矩阵内积运算
c * d
print(c * d)
```

----矩阵的内积运算

- 数组的乘法是对应元素相乘，不是矩阵内积，矩阵内积使用的是np.dot()函数

```
# 创建一个3×2的数组矩阵，每个元素是1~5的随机整数，random_integers函数包含末尾数5
a = np.random.random_integers(1 , 5 , (3 , 2))
print(a)
# 创建一个2×3的数组矩阵，每个元素是1~5的随机整数
b = np.random.random_integers(1 , 5 , (2 , 3))
print(b)
# 矩阵内积
np.dot(a, b)
print(np.dot(a, b))
```

----数组的比较运算

- 数组可以直接进行比较，返回一个同维度的布尔数组。针对布尔数组，可以使用all()/any()函数来返回布尔数组的标量值

```
a= np.array([1 , 2 , 3 , 4])
b = np.array([4 , 2 , 2 , 4])
print(a == b)
print(a > b)
print((a==b).all())   # 必须布尔矩阵中所有元素均为True时结果才为True
print((a== b).any())  # 只要布尔矩阵中有任一元素为True时结果就为True
```

----数组的内置函数

```
a = np.arange(6)
print(a)

# 求余弦
print(np.cos(a))

# 求指数
print(np.exp(a))

# 求平方根
print(np.sqrt(a))
```

----数组的统计运算

（1）一维数组

```
# 创建6个元素的一维向量，元素为1~5之间随机整数
a= np.random.random_integers(1, 5 , 6)
print(a)

# 求和
print(a.sum())

# 求平均值
print(a.mean())

# 求标准差
print(a.std())
```

```
# 求最小值
print(a.min ())

# 求最大值
print(a.max())

# 最小值元素所在的索引
print(a.argmin())

# 最大值元素所在的索引
print(a.argmax())
```

（2）二维数组

- 针对二维数组或者更高维度的数组，可以根据行或列来计算。axis 参数表示坐标轴， 0 表示按列计算， 1表示按行计算。需要特别注意的是，按列计算后， Numpy 会将计算结果默认转换为行向量

```
# 创建一个6×4二维数组矩阵，每个元素的值为1~5之间的随机整数
b = np.random.random_integers(1 , 5 , (6 , 4))
print(b)

# 求矩阵中所有元素的和
print(b.sum())

# 按照矩阵的列求和，axis=0表示按列
print(b.sum(axis=0))

# 按照矩阵的行求和，axis=1表示按行
print(b.sum(axis=1))
```

```
# 先按照行求和，得到一个数组，再将数组的所有元素求和
print(b.sum(axis=1).sum())

# 按照矩阵的行求每一行的最小值
print(b.min(axis=1))

# 按照矩阵的行求每一行的最小值所对应的索引值（指的是在特定行的索引）
print(b.argmin(axis=1))

# 按照矩阵的行求每一行的标准差
print(b.std(axis=1))
```

- 数组的维度变换

- reshape()：转换维度，变成多维数组
- ravel()：将多维数组“摊平”，变成一维向量

```
# 创建一维向量，元素为1~11的12个整数
a= np.arange(12)
print(a)

# 转换为4×3 的二维数组
b = a.reshape(4 , 3)
print(b)

# 变为一维向量，转换时按照行的顺序进行排列，第2行接在第1行的末尾
print(b.ravel())
```

```
# 创建一维向量，元素为1~3的4个整数
a= np.arange(4)
print(a)
# 打印纬度信息，为(4,)，即只有第1维具有4个元素，没有第2维
print(a.shape)

# 在列上添加一个维度，纬度信息为(4, 1)，变成4×l数组，其自身位于第1列
b = a[:, np.newaxis]
print(b)
# 打印纬度信息，为(4, 1)，即4×1的数组
print(b.shape)

# 在行上添加一个维度，纬度信息为(1, 4)，变成l×4数组，其自身位于第1行
c = a[np.newaxis,:]
print(c)
# 打印纬度信息，为(1, 4)，即l×4的数组
print(c.shape)
```

- 数组的排序及索引

```
# 创建6×4的二维数组，元素为1~10的10个随机整数
a = np.random.random_integers(1, 10 , (6, 4))
print(a)

# 按行独立排序，返回一个备份b，即a不变
b = np.sort(a, axis=1)
print(b)

# 按列排序，直接把结果保存到当前数组
a.sort(axis=0)
print(a)

# 创建由6个1~10的随机整数构成的一维向量
a = np.random.random_integers(1 , 10 , 6)
# 打印排序前的a
print(a)
```

```
# 此时的a仍然是排序前的数组
idx = a.argsort()
# 打印排序前的a
print(a)
# 假如将a排序后，a中各个元素在排序前的索引值
print(idx)

# a仍然是原始的排序前的a，而a[idx]是一个虚拟的排序后的a
print(a[idx])
# 将虚拟的排序后的数组，赋值给b，使得b成为物理上排序后的数组
b = a[idx]
# a仍然是排序前的数组，而b是排序后的数组
print(b)
# 由于a并没有物理上的排序操作，因此a仍然是排序前的a，下面打印出排序前的a
print(a)
```

- 数组的文件操作（数据文件存储、数据文件导入）
  - Numpy数组作为文本文件，可以直接保存到文件系统里，也可以从文件系统里读取出数据，也可以直接保存为Numpy 特有的二进制格式

```
a = np.arange(15).reshape(3 , 5)
# 将数组保存文本文件
np.savetxt("test.txt", a)  # 保存时将整型转换成浮点型

# 读取文本文件，恢复成数组
b = np.loadtxt("test.txt")
print(b)

# 将数组保存为二进制文件
np.save("test.npy", a)

# 读取二进制文件，恢复成数组
c = np.load("test.npy")
print(c)
```

• Pandas

Pandas 是一个强大的时间序列数据处理工具包，最初开发的目的是为了分析财经数据，现在已经广泛应用在 Python 数据分析领域中

Pandas 的基本数据结构：

• Series 数据结构：行数据，一维数组

• DataFrame 数据结构：行列数据，二维数组，每一行或者列都是 Series 结构。DataFrame 里的数据实际是用 Numpy 的 array 对象来保存的。

4、pandas：

（1）数据映射

```
# 创建一个 Series 对象 a
a = pd.Series(['Java','C','C++'])
print(a)
0  Java
1  C
2  C++
dtype: object
#对 a 中的数据进行映射转换
b= a.map({'Java': '11','C': '22','C++': '33'})
print('b 组：')
print(b)
0   11
1   22
2   33
dtype: object
```

（2）数据排序

```
#通过 DataFrame.sort_index()函数对索引进行排序，
#通过 DataFrame.soti_values()对数值进行排序
# DataFrame 是二维数组对象
df = pd.DataFrame(np.random.randn(6,4), columns = list('ABCD'))
print(df)
          A         B         C         D
0  0.329717 -0.059489  0.901134 -0.859686
1 -0.301952  0.001746  3.621898  1.172631
2 -1.241323 -0.162597 -1.105483  0.024316
3  1.196289 -1.040562 -1.001477 -1.553270
4  1.747654 -0.778663  0.730327 -0.427326
5 -1.475357  0.044950 -0.387159 -1.998673
# 根据列名称进行逆序排列，axis=1 表示按列
print(df.sort_index(axis=1, ascending=False))
```

结果为：按 D C B A 顺序输出

```
#根据 B 这一列的数据从小到大进行排序
print(df.sort_values(by ='B'))
```

（3）数据访问

- 根据行索引，或者列名称来访问多个数据，或者通过数组索引访问特定元素

```
# DataFrame是二维数组对象
df = pd.DataFrame(np.random.randn(6 , 4) , columns = list('ABCD'))
print(df)

# 获取原始数据，不包含行号、列名称，只包含纯数据
print(df.values)

# 通过行索引范围来访问特定几行的数据
print(df[3 : 5])

# 选择A 、B 、D 这3 列数据:
print(df[['A','B','D']])

# 通过标签来选择某个元素
print(df.loc[3 ,'A'])
```

```
# 通过数组索引来访问某个元素
print(df.iloc[3 , 0])
print(df.iloc[2 : 5 , 0 : 2])

# 选择C列里所有大于0的数据所在的行
print(df[df.C > 0])

# 添加一列，列名为TAG
df['TAG']=['cat', 'dog', 'cat', 'cat', 'cat', 'dog']
print(df)

# 根据TAG列做分组统计。
print(df.groupby('TAG').sum())
```

（4）文件操作：数据文件保存、导入

#使用 DataFrame.to_csv()函数把数据保存到文件中，

#使用 DataFrame.read_csv()函数从文件中读取数据

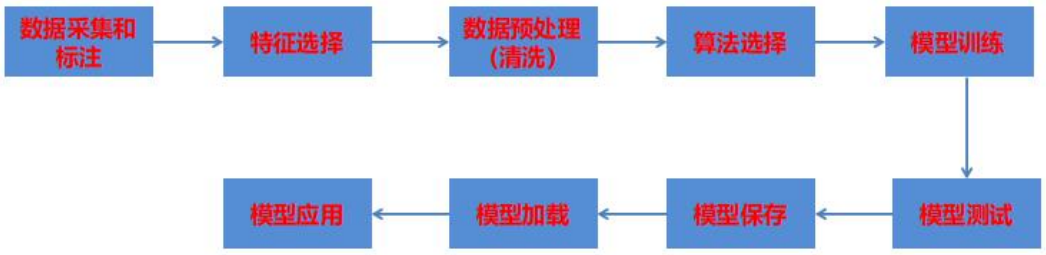
```
# 读取文件，并且从第0列开始读取
df = pd.read_csv('data.csv', index_col=0)

# 数据的形状，即(行数, 列数)，如(100, 4)
print(df.shape)

# 数据的前5行
print(df.head(5))

# 保存数据到csv文件
df.to_csv( 'csv_data_save.csv')

# 保存数据到excel文件
df.to_excel( 'excel_data_save.xls')
```

5、Matplotlib

Matplotlib 是 Python 数据可视化工具包

四、机器学习框架 Scikit-Learn

1、机器学习流程



2、数据预处理

数据预处理工具：Numpy，Pandas

3、数据标准化

- 数据正规化：sklearn.preprocessing.Normalizer(norm='l2')

- 正则化的过程是将每个样本缩放到单位范数(每个样本的范数为 1)，对每个样本计算其 p-范数，然后对该样本中每个元素除以该范数，这样处理的结果是使得每个处理后样本的 p-范数(L1-norm,L2-norm)等于 1。

- 数据归一化（零均值）：sklearn.preprocessing.StandardScaler()

- 将数据按其属性(按列进行)减去其均值，然后除以其方差。最后得到的结果是，对每个属性/每列来说所有数据都聚集在 0 附近，方差值为 1。

- 最大最小标准化：sklearn.preprocessing.MinMaxScaler(feature_range=(0, 1))

- 将属性缩放到一个指定的最大值和最小值(通常是 1-0)之间。

4、机器学习方法

- 有监督学习
- 无监督学习
- 半监督学习

5、有监督学习

- 回归
- 分类

很多算法既可以用于分类，也可以用于回归，如：决策树、支持向量机、随机森林、Adaboost 等

（1）回归

- 线性回归

线性回归：sklearn.linear_model.LinearRegression()

- 没什么特别参数，正常使用默认参数

- 多项式变换

多项式特征变换后再线性回归，本质上是增加特征数量，构造新的特征以解决欠拟合问题

多项式变换：sklearn.preprocessing.PolynomialFeatures(degree=2)

- 主要参数是 degree，体现了变换的复杂性

示例代码：

```
quadratic = PolynomialFeatures(degree=2)
X_train = quadratic.fit_transform(X_train)
X_test = quadratic.transform(X_test)
```

（2）• 分类

- 逻辑回归：sklearn.linear_model.LogisticRegression(penalty='l2', C=1.0)

主要参数：

Penalty: {‘l1’, ‘l2’, ‘elasticnet’, ‘none’}, default=’l2’。正则化方式。

C: float, default=1.0。正则化强度的倒数；必须是正浮点数。

- 决策树（随机森林的特例）：sklearn.tree.DecisionTreeClassifier()
- 朴素贝叶斯：sklearn.naive_bayes()。朴素贝叶斯算法在自然语言领域有广泛的应用
  - 伯努利分布：skleam.naive_bayes.BemoulliNB()
  - 多项式分布：skleam.naive_bayes.MultinomialNB()
  - 高斯分布：skleam.naive_bayes.GaussianNB()
- K 近邻（KNN）：sklearn.neighbors.KNeighborsClassifier(n_neighbors=k)
- 支持向量机（SVM）：sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3)。

Kernel 参数：kernel：kernel{‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’},

核函数的本质：线性不可分情况下，将数据进行非线性映射到高维空间，高维特征空间中变成线性可分

- 集成学习：

集成算法可以有效地解决过拟合问题

（1）随机森林：sklearn.ensemble.RandomForestClassifier()；

以决策树作为基分类器（学习器）。

（2）AdaBoost：sklearn.ensemble.AdaBoostClassifier()

默认以决策树作为基分类器（学习器）。

6、无监督学习

无监督学习：

- 聚类
- 降维

（1）聚类

- K 均值聚类（K-Means）：sklearn.cluster.KMeans(n_clusters=8)

  主要参数：n_clusters：int, default=8。聚类的数量
- 凝聚型层次聚类（agglomerative）：sklearn.cluster.AgglomerativeClustering(n_clusters=2)

（2）降维

- PCA：sklearn.decomposition.PCA()

使用方法：

```
model = sklearn.decomposition.PCA(n_components) # 创建 PCA 模型
model.fit_transform(X, y=None) # 对数据集 X 进行 PCA 降维操作，返回值为降维后的数据集
```

- LDA：sklearn.discriminant_analysis.LinearDiscriminantAnalysis()

使用方法：

```
# 创建 LDA 模型
model = sklearn.discriminant_analysis.LinearDiscriminantAnalysis(solver, n_components)
# 对数据集 X 进行 LDA 降维操作，返回值为降维后的数据集
model.fit_transform(X, y=None)
```

7、训练

数据集划分方法：通常将数据集的 80%作为训练集，20%作为测试集

训练方法：fit()方法。每一种算法都有相应的 fit()方法，如 SVM 算法：sklearn.svm.SVC.fit(train_X, train_y)，得到训练完成的模型，此时模型参数已经生成

两种训练中出现的现象：

过拟合：模型过于复杂，过度的拟合了训练数据。解决方案：获取更多的训练数据、减少输入的特征数量、降低模型复杂度，如减少网络深度等

欠拟合：模型过于简单，连训练数据都无法很好的拟合。解决方案：增加有价值的特征、增加模型的复杂性，如增加多项式特征，增加网络深度等

8、预测

sklearn 的 predict()方法

如 SVM 算法：sklearn.svm.SVC.predict(test_X)，返回值是预测的结果

9、参数调优

超参数：不是由算法训练出来的，而是人为设定的，不同算法有不同的超参数

不同算法的超参数举例：

KNN：K 参数

多项式回归：多项式特征变换中的 degree 参数

决策树：树的最大深度 max_depth 参数

支持向量机：惩罚系数 C 参数、核函数种类 kernel 参数、 gamma 参数。

10、模型选择： 交叉验证

交叉验证法是一种超参数选择的方法，也叫模型选择，主要用于超参数调优，得分最佳的超参数为最优超参数，进行交叉验证时需要用到验证集。当数据集比较小的时候，交叉验证可以“充分利用”有限的数据找到合适的模型参数，防止过度拟合

交叉验证的使用方法：sklearn.model_selection.cross_val_score()，

11、模型评估

（1）• 分类：sklearn.metrics.classification_report()

返回值为模型综合评估报告(包含了准确率 Precision、精确度 accuracy、召回率 recall、f1-score)

（1）准确率（Precision）：查准率

sklearn.metrics.precision_score()，返回值为模型的 precision 得分

Precision 和 Recall 都是针对某个类而言的，例如：正样本的 Precision 表示预测为正的样本中有多少预测对了

（2）精确度（accuracy ）

sklearn.metrics. accuracy_score() ，返回值为模型的 accuracy 得分

Accuracy 表示在所有样本中，有多少比例的样本被预测对了

（3）召回率（Recall）：查全率

sklearn.metrics.recall_score()，返回值为模型的 recall 得分

例如：正样本的 Recall 表示真实标签为正的样本有多少被预测对了

（4）F1 Score

sklearn.metrics.f1_score() ，返回值为模型的 f1 得分

（2）• 聚类：sklearn.metrics.silhouette_score()

返回值为所有样本的平均 silhouette 系数

（3）• 回归：sklearn.metrics.r2_score()

返回值为 R2 得分

12、模型应用

• 模型保存：joblib.dump()：将模型对象保存到文件

主要参数：

model：模型对象

filename：模型文件名，含路径

- 模型加载：joblib.load()：返回值为模型对象

  主要参数：

  filename：模型文件名，含路径

五、深度学习框架 Tensorflow2

1、TensorFlow 支持 CPU 和 GPU 作为计算资源。GPU 运行速度快，需要 NVIDA 运行库支持

运行库版本：　CUDA 10.0　CuDNN 7.5.0

在 TensorFlow 2.0 中所有的高阶 API 全部集中到 tf.keras 库下

2、Tensorflow2.0 开发流程

3、Tensorflow 基本数据类型

- Tensor 的概念
- Tensor 的基本运算
- 低阶 API
- 高阶 API

4、Tensor 的概念

TensorFlow 中的 Tensor，翻译为中文就是张量，TensorFlow 就是流动的张量。

张量（Tensor）：是具有统一类型（称为 dtype）的多维数组。所有张量都是不可变的，永远无法更新张量的内容，只能创建新的张量，张量有最重要的 2 个属性：

形状 shape：张量的每个维度（轴）的长度（元素数量），0 维、1 维、2 维、3 维等

数据类型 dtype：float32、int32、或者 string 等

（1）形状：0 维

0D 张量只包含一个数字，有 0 个维度，又称为标量

示例代码：

```
rank_0_tensor = tf.constant(4)
print(rank_0_tensor)
```

运行结果：

```
tf.Tensor(4, shape=(), dtype=int32)
```

（2）1 维

1D 张量包含一个一维数组，可以看作由 0D 张量组成的数组，有 1 个维度，又称为向量

示例代码：

```
rank_1_tensor = tf.constant([2.0, 3.0, 4.0])
print(rank_1_tensor)
```

运行结果：

tf.Tensor([2. 3. 4.], shape=(3,), dtype=float32)

（3）2 维

2D 张量可以看作由 1D 张量组成的数组，有 2 个维度，又称为矩阵

示例代码：

```
rank_2_tensor = tf.constant([[1, 2],
                             [3, 4],
                             [5, 6]], dtype=tf.float16)
print(rank_2_tensor)
```

运行结果：

```
tf.Tensor(
    [[1. 2.]
    [3. 4.]
    [5. 6.]], shape=(3, 2), dtype=float16)
```

（4）3 维

3D 张量可以看作由 2D 张量组成的数组，有 3 个维度

示例代码：

```
rank_3_tensor = tf.constant([
    [ [0, 1, 2, 3, 4],
    [5, 6, 7, 8, 9] ],

    [ [10, 11, 12, 13, 14],
    [15, 16, 17, 18, 19]],
    [[20, 21, 22, 23, 24],
    [25, 26, 27, 28, 29] ],      ])
print(rank_3_tensor)
```

运行结果：

```
tf.Tensor(
    [[[ 0 1 2 3 4]
    [ 5 6 7 8 9]]
    [[10 11 12 13 14]
    [15 16 17 18 19]]
    [[20 21 22 23 24]
    [25 26 27 28 29]]], shape=(3, 2, 5), dtype=int32)
```

5、Tensor 的基本运算

基本运算名称：

- 标量运算
- 向量运算
- 矩阵运算

（1）标量运算

包括加、减、乘、除、乘方以及三角函数、指数、对数等常见函数

示例代码：

```
a = tf.constant([[1, 2], [3, 4]])
b = tf.constant([[0, 0], [1, 0]])
print(a + b)    tf.Tensor( [[1 2] [4 4]], shape=(2, 2), dtype=int32)
print(a - b)    tf.Tensor( [[1 2] [2 4]], shape=(2, 2), dtype=int32)
```

print(a * b)　　tf.Tensor([[0 0] [3 0]], shape=(2, 2), dtype=int32)

print(a / b)　　tf.Tensor([[inf inf] [3. inf]], shape=(2, 2), dtype=float64)

（2）向量运算

示例代码：

```
A = tf.constant([[2, 20, 30, 3, 6],
                 [1, 1, 1, 1, 1]])
print(tf.math.reduce_sum(A))    运行结果：tf.Tensor(66, shape=(), dtype=int32)
print(tf.math.reduce_max(A))    运行结果：tf.Tensor(30, shape=(), dtype=int32)
B = tf.constant([[2, 20, 30, 3, 6],
                 [3, 11, 16, 1, 8],
                 [14, 45, 23, 5, 27]])
print(tf.math.reduce_sum(B, 0)) # 沿 0 轴（列方向），求和
tf.Tensor([19 76 69 9 41], shape=(5,), dtype=int32)
print(tf.math.reduce_sum(B, 1)) # 沿 1 轴（行方向），求和
tf.Tensor([ 61 39 114], shape=(3,), dtype=int32)
print(tf.math.reduce_max(B, 0)) # 沿 0 轴，求最大值
tf.Tensor([14 45 30 5 27], shape=(5,), dtype=int32)
print(tf.math.reduce_max(B, 1)) # 沿 1 轴，求最大值
tf.Tensor([30 16 45], shape=(3,), dtype=int32)
```

（3）矩阵运算

矩阵运算：矩阵必须是二维的，包括矩阵乘法、矩阵转置、矩阵逆、矩阵行列式、矩阵求特征值、矩阵分解等运算

示例代码：

```
A= tf.constant([[2,20,30,3,6],
                [1,1,1,1,1]])

#矩阵转置
A_trans = tf.linalg.matrix_transpose(A)
print(A_trans)
tf.Tensor( [[ 2 1] [20 1] [30 1] [ 3 1] [ 6 1]], shape=(5, 2), dtype=int32)
B= tf.constant([[2,20,30,3,6],
                [3,11,16,1,8],
                [14,45,23,5,27]])
#矩阵点积
print(tf.linalg.matmul(B, A_trans))
tf.Tensor( [[1349 61] [ 757 39] [1795 114]], shape=(3, 2), dtype=int32)
```

6、低阶 API

- tf.constant()：提供了常量的声明功能

  示例代码：

  ```
  a = tf.constant(7)
  print(a)        tf.Tensor(7, shape=(), dtype=int32)
  print(a.numpy())        7
  ```

- tf.Variable()：提供了变量的声明功能

  ```
  #声明一个 Python 变量
  ```

```
a1 = 7
print('a1= ',a1)
a1= 7
#声明一个 0 阶 Tensor 变量
a2 = tf.Variable(7)
print('a2= ',a2)
a2= <tf.Variable 'Variable:0' shape=() dtype=int32, numpy=7>
#声明一个 1 阶 Tensor 变量，即数组
a3 = tf.Variable([0,1,2])
print('a3= ',a3)
a3= <tf.Variable 'Variable:0' shape=(3,) dtype=int32, numpy=array([0, 1, 2])>
print(a1,a2,a3)
7 <tf.Variable 'Variable:0' shape=() dtype=int32, numpy=7> <tf.Variable
'Variable:0' shape=(3,) dtype=int32, numpy=array([0, 1, 2])>
```

• tf.reshape()：提供了多阶 Tensor 的形状变换功能

```
a = tf.Variable([[0,1,2],[3,4,5]])
print(a)
<tf.Variable 'Variable:0' shape=(2, 3) dtype=int32,
numpy= array([[0, 1, 2], [3, 4, 5]])>
# 对 a 的形状进行变换，变换为(3,2)
a1 = tf.reshape(a,[3,2])
print(a1)
tf.Tensor( [[0 1] [2 3] [4 5]], shape=(3, 2), dtype=int32)
print(a1.shape)
(3, 2)
```

• tf.math.reduce_mean()：提供了对 Tensor 求平均值的功能

```
a = tf.constant([1,2.,3,4,5,6,7.])
#输入数据类型是 float32，输出数据类型也是 float32
print(a.dtype)
<dtype: 'float32'>
print(tf.math.reduce_mean(a))
tf.Tensor(4.0, shape=(), dtype=float32)
b = tf.constant([[1,2,1],[5,2,10]])
#输入数据类型是 int32，输出数据类型也是 int32
print(b.dtype)
<dtype: 'int32'>
#虽然平均值为 3.5，但是由于上面确定了输出类型为整型，因此强制赋值为整数 3
print(tf.math.reduce_mean(b))
tf.Tensor(3, shape=(), dtype=int32)
```

• tf.random.normal()：随机生成一个 Tensor，其值符合正态分布

```
#shape：Tensor 的维度  mean：正态分布的中心值
a = tf.random.normal(shape=[2,3], mean=2)
print(a)# a 为 Tensor 类型，是一个 2 维张量 Tensor
print(type(a))# type()方法是 Python 的原生方法，查看 Tensor 对象 a 的类型，是一个 Tensor
```

```
print(a.dtype)#由于 a 是 Tensor 类型，因此可以使用 Tensor 对象的 dtype 属性
print(a.numpy())# a 转换成 numpy 的 array 数组类型，就是普通的数组类型
print(type(a.numpy()))
```

```
tf.Tensor(
[[3.093015   0.83826673 0.6251465 ]
 [3.308845   3.5064597  2.4058805 ]], shape=(2, 3),
dtype=float32)
<class 'tensorflow.python.framework.ops.EagerTensor'>
<dtype: 'float32'>
[[3.093015   0.83826673 0.6251465 ]
 [3.308845   3.5064597  2.4058805 ]]
<class 'numpy.ndarray'>
```

• tf.random.uniform()：随机生成一个 Tensor，其值符合均匀分布

```
a = tf.random.uniform(shape=[2,3],minval=1,maxval=10,seed=8,dtype=tf.int32)
print(a.numpy)
```

```
<bound method _EagerTensorBase.numpy of <tf.Tensor:
shape=(2, 3), dtype=int32, numpy=
array([[4, 1, 4],
       [2, 5, 9]])>>
```

```
print(a.numpy())
[[4 1 4]
 [2 5 9]]
```

• tf.transpose()：提供了矩阵的转置功能

```
#定义 x 为一个 3 维张量，形状是(2,2,3)
x = tf.constant([[[1,2,3],
                  [4,5,6]],
                 [[7,8,9],
                  [10,11,12]]])
#转置后的形状是(2,3,2)
a = tf.transpose(x,perm=[0,2,1])
print(a.numpy())
```

```
[[[ 1  4]
  [ 2  5]
  [ 3  6]]

 [[ 7 10]
  [ 8 11]
  [ 9 12]]]
```

• tf.math.argmax()：提供了返回一个数组内最大值对应索引的功能

```
a = tf.constant([1,2,3,4,5])
x = tf.math.argmax(a)
print(x.numpy())      4
```

• tf.expand_dims()：在输入的 Tensor 中增加一个维度

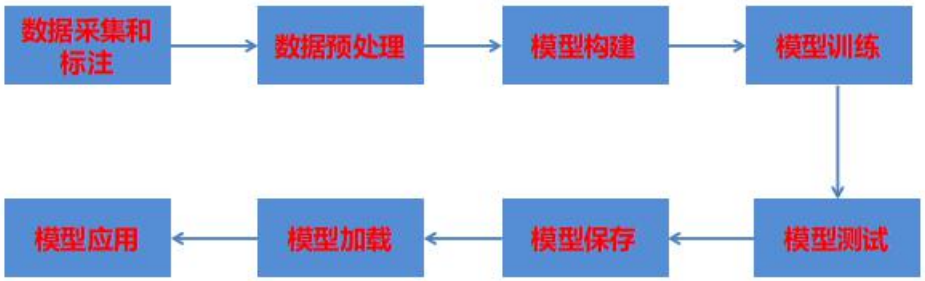
- tf.concat()：将多个 Tensor 在同一个维度上进行连接
- tf.bitcast()：提供了数据类型转换功能

7、Tensorflow 高阶 API（tf.keras）

使用方法为 tf.keras.API。在 TensorFlow 2.0 版本中，官方推荐的高阶 API 只有 tf.keras。

- activations：tf.keras.activations 中包含了当前主流的激活函数（目的：非线性变换）
- applications：tf.keras.applications 中包含的是已经进行预训练的神经网络模型，可以直接进行预测或者迁移学习
- backend：tf.keras.backend 中包含了 Keras 后台的一些基础 API 接口
- datasets：tf.keras.datasets 中包含了常用的公开数据训练集
- layers：tf.keras.layers 中包含了已经定义好的常用的神经网络层（卷积层、池化层、全连接层、激活层、正则化层、Dropout 层等）
- losses：tf.keras.losses 中包含了常用的损失函数
- optimizers：tf.keras.optimizers 中包含了主流的优化器（求取模型的最优训练参数，注意不是超参数）
- preprocessing：tf.keras.preprocessing 中包含了数据处理的一些方法
- regularizers：tf.keras.regularizers 中提供了常用的正则化方法（防止过拟合），包括 L1、L2 等正则化方法
- Sequential：tf.keras.Sequential 将神经网络层进行线性组合形成神经网络结构。Sequential 是一个方法类，可以轻而易举地以堆叠神经网络层的方式集成构建一个复杂的神经网络模型。Sequential 提供了丰富的方法，利用这些方法可以快速地实现神经网络模型的网络层级集成、神经网络模型编译、神经网络模型训练和保存，以及神经网络模型加载和预测

8、深度学习的基本流程 无需进行特征选择



9、Tensorflow 关键步骤（1）

- 模型搭建：两种模型搭建的方法

（1）Sequential 编程方法

（2）函数式 API 编程方法

- 模型创建：model = tf.keras.Sequential()
- 模型编译：tf.keras.Sequential().compile()，即：model.compile()。只有编译后才能对神经网络模型进行训练。对神经网络模型进行编译是将高阶 API 转换成可以直接运行的低阶 API，理解时可以类比高级开发语言的编译
- 模型训练：tf.keras.Sequential.fit()，即：model.fit()
- 模型保存：tf.keras.Sequential.save()，即：model.save()
- 模型加载：model = tf.keras.models.load_model()
- 模型预测：tf.keras.Sequential.predit()，即：model.predict()

（2）Tensorflow 关键步骤（2）

在模型搭建中，有两种主要方法：Sequential 方法、函数式 API 方法：

• Sequential 方式：顺序式编程，使用 add()方法添加网络层。该方法适用于简单的网络，属于低级编程模式

• 函数式 API 方式：自由度更大的编程方式，使用 API 函数添加网络层，适用于复杂网络构建，属于高级编程模式，能够实现 Sequential 方式无法胜任的复杂网络构建

10、模型创建(Sequential 方法)：

（1）实例化模型对象

首先要创建一个模型对象的实例，方法是：tf.keras.Sequential()，返回值为模型的实例 model

示例代码：

```
# 创建模型对象的实例
model = tf.keras.Sequential()
```

（2）组件网络层

使用 Sequential().add()方法来实现神经网络层级的集成，可以根据实际需要将 tf.keras.layers 中的各类神经网络层级添加进去。

示例代码：实现三个全连接神经网络层级的集成，构建一个全连接神经网络模型

```
import tensorflow as tf
# 创建模型对象的实例
model = tf.keras.Sequential()
# 使用 add()方法集成神经网络层级
model.add(tf.keras.layers.Dense(256, activation="relu"))
model.add(tf.keras.layers.Dense(128, activation="relu"))
model.add(tf.keras.layers.Dense(2, activation="softmax"))
```

11、模型编译

Sequential().compile()：提供了神经网络模型的编译功能，需要定义三个参数：

loss：用来配置模型的损失函数，可以通过名称调用 tf.losses API 中已经定义好的 loss 函数

optimizer：用来配置模型的优化器，可以调用 tf.keras.optimizers API 配置模型所需要的优化器

metrics：用来配置模型评价的方法，如 accuracy、mse 等

示例代码：

```
# 编译模型
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(0.01),
              metric=["accuracy"])
```

12、模型训练

神经网络模型编译后，可以使用准备好的训练数据对模型进行训练

Sequential().fit()：提供了神经网络模型的训练功能，主要的配置参数如下：

x：配置训练的输入数据，可以是 array 或者 tensor 类型

y：配置训练的标注数据，可以是 array 或者 tensor 类型

batch_size：配置批大小，默认值是 32 数据分成几批

epochs：配置训练的 epochs 的数量 训练集被用几次

13、模型保存

使用 save()或者 save_weights()方法保存并导出训练得到的模型，在使用这两个方法时需

要分别配置以下参数：

save()方法的参数配置	save_weights()方法的参数配置
• filepath：配置模型文件保存的路径。 • overwrite：配置是否覆盖重名的 HDF5 文件。 • include_optimizer：配置是否保存优化器的参数。	• filepath：配置模型文件保存的路径。 • overwrite：配置是否覆盖重名的模型文件。 • save_format：配置保存文件的格式。

14、模型加载和预测

当需要使用模型进行预测时，可以使用 tf.keras.models 中的 load_model()方法重新加载已经保存的模型文件。在完成模型文件的重新加载之后，可以使用 predict()方法对数据进行预存输出。在使用这两个方法时需要分别进行如下参数配置：

load_model()方法的参数配置	predict()方法的参数配置
• filepath：加载模型文件的路径。 • custom_objects：配置神经网络模型自定义的对象。如果自定义了神经网络层级，则需要进行配置，否则在加载时会出现无法找到自定义对象的错误。 • compile：配置加载模型之后是否需要进行重新编译。	• x：配置需要预测的数据集，可以是 Array 或者 Tensor。 • batch_size：配置预测时的批大小，默认值是 32。

示例代码：

```
# 模型加载
model = tf.keras.models.load_model(filepath)
# 模型预测
model.predict(x)
```

15、Python 图像处理工具 PIL

PIL 是 Python Imaging Library 的简称，是 Python 生态系统中图像处理的标准库，其功能非常强大且 API 非常简单、易用，因此深受欢迎。除了 PIL 之外，常用的图像处理工具是 OpenCV 库。

PIL 的基本功能：

图像读写

图像编辑

（1）图像读写

使用 Image 类进行图像的读写：

- 读取图像：Image.open()：提供了打开图像文件和读取图像数据的功能

  示例代码：

  ```
  from PIL import Image
  with open("test.jpg", "rb") as fp:
    im = Image.open(fp) # 打开图片文件，将图片数据保存到 im 对象中去
  ```

- 图像保存：Image.save()：提供了图像数据的保存功能，可以保存成训练所需要的图像格式

  示例代码：

  ```
  from PIL import Image
  import os, sys
  infile = "test.jpg"
  ```

```
f, e = os.path.splitext(infile)
outfile = f + ".png"
if infile != outfile:
  try:
  Image.open(infile).save(outfile)
  except IOError:
  print("cannot convert", infile)
```

（2）图像编辑

使用 Image 类进行图像的编辑：

- 图像截取：crop()
- 图像尺寸变换：resize()
- 像素变换：convert()

- 图像截取：crop()：提供了对图像进行截取以保持图像的尺寸统一的功能。

示例代码：

```
from PIL import Image
# 读取图像数据
im = Image.open("test.jpg")
# 初始化截取图像的范围
box = (100, 100, 400, 400)
# 截取的范围是：左上角坐标(100,100)，右下角坐标(400,400)，截取矩形
# 完成图像的截取并保存图像
im.crop(box).save("crop.jpeg")
```

- 图像尺寸变换：resize()：提供了图像尺寸变换功能，可以按照需要变换源图像的尺寸。

示例代码：

```
from PIL import Image
# 读取图像数据
im = Image.open("test.jpg")
# 改变图片尺寸为(宽,高)，并保存
im.resize((500, 400)).save("resize.jpeg")
```

- 像素变换：convert()：对图像进行二值化处理。这个 API 提供了将图像进行像素色彩模式转换的功能，可以在支持的像素色彩格式间进行转换。在人工智能算法编程中常用的是将 RGB 模式进行二值化操作。

示例代码：

```
from PIL import Image
# 读取图像数据
im = Image.open("test.jpg")
# 将彩色图片转换成灰度图像
im.convert("L").save("convert.jpeg")
```

16、CIFAR-10 数据集分析说明

17、Tensorflow 构建卷积神经网络（CNN）

卷积神经网络（CNN）主要用于计算机视觉，如图像、视频类的应用

主要有以下的层次构成：

- 数据输入层：主要是对原始图像数据进行预处理。预处理方式：去均值、归一化、PCA

- 卷积层：卷积层通过卷积计算将样本数据进行降维采样，以获取具有空间关系特征的数据
- 激活层：激活层对数据进行非线性变换处理，目的是对数据维度进行扭曲来获得更多连续的概率密度空间。在 CNN 中，激活层一般采用的激活函数是 ReLU，它具有收敛快、求梯度简单等特点
- 池化层：池化层夹在连续的卷积层中间，用于压缩数据的维度以减少过拟合
- 压平层：tf.keras.layers.Flatten()：Flatten 将输入该层级的数据压平，不管输入数据的维度数是多少，都会被压平成一维
- 全连接层：tf.keras.layer.Dense()：Dense 提供了全连接的标准神经网络。和常规的 DNN 一样，全连接层在所有的神经网络层级之间都有权重连接，最终连接到输出层。在进行模型训练时，神经网络会自动调整层级之间的权重以达到拟合数据的目的
- Dropout 层（失活）：Dropout 的工作机制就是每步训练时按照一定的概率随机使神经网络的神经元失效，这样可以极大降低连接的复杂度。同时由于每次训练都是由不同的神经元协同工作的，这样的机制也可以很好地避免数据带来的过拟合，提高了神经网络的泛化性
- Adam：Adam 是一种可以替代传统随机梯度下降算法的梯度优化算法

18、Tensorflow 构建模型的两种方法

- Sequential 方式：顺序式编程，使用 add()方法添加网络层。该方法适用于简单的网络，属于低级编程模式。
- 函数式 API 方式：自由度更大的编程方式，使用 API 函数添加网络层，适用于复杂网络构建，属于高级编程模式，能够实现 Sequential 方式无法胜任的复杂网络构建