## 2.5.2 图像分类器、识别器与机器学习模型网络实验应用平台部署

该部分的目的是阐述如何将机器学习模型部署到网络平台上，同时包括信息的收集，因此主要包括两大部分内容，一是，应用Flask构建网络应用平台，这包括网络平台的基本架构、问卷调研和将机器学习模型图像识别器嵌入到网络中；二是，构建图像分类器和识别器，这将涉及到机器学习相关内容下的分类模型决策树（Decision trees）和随机森林（Random forests），而作为学习的样本特征的提取则使用了视觉词袋的方法来构建图像映射特征用于模型的训练。

### 2.5.2.1 用Flask构建实验用网络应用平台

在很多场景中，都需要借助网络完成相关任务，例如展示研究内容，开展问卷调查收集数据，提供服务（例如机器学习或深度学习中已训练好的模型在线预测等）。因为这些任务需要更多的自由性，例如读写和处理数据，也需要根据不同的任务调整网页内容，因此使用类似[WIX](https://www.wix.com/)①快速网页构建服务提供的方式很难满足数据分析、可视化、模型预测服务及布局调研信息等方面要求，那么[Flask](https://pypi.org/project/Flask/)②成为空间数据分析研究最好的网络实验构建平台。一方面，Flask是应用Python语言编写，是数据分析、大数据分析、机器学习和深度学习广泛使用的语言；同时，Flask是一个轻量级的可定制框架，灵活、轻便、安全，容易上手，能够快速的学习并实现自行搭建网络平台完成相关网络实验部署；Flask也并不限制使用何种数据库，何种模板样式，具有强劲的自由拓展性，实现自由的定制需求。

学习Flask推荐阅读[官方文档](https://flask.palletsprojects.com/en/1.1.x/)③，及教材*Flask Web Development: Developing Web Applications with Python（FWD）*[1]。城市空间数据分析网络实验平台的建设，是以上述教材提供的案例为基础（包含社交博客搭建），在此之上扩展不同的实验任务，网络实验的内容及架构代码位于GitHub上[caDesign\_ExperimentPlatform](https://github.com/richieBao/caDesign_ExperimentPlatform)④代码仓库中。

Flask目前已经完全的整合进了[PyCharm](https://www.jetbrains.com/pycharm/)⑤，因此推荐使用Pycharm构建Flask网络实验平台，无需自行搭建Flask的Python环境。同时对于网页的开发，Pycharm提供了非常友好的书写代码的环境，能够节约研究者搭建的时间。FWD教材提供的案例电子邮件传输协议（SMTP）基于Google Gmail，如果在中国可以使用QQ邮箱提供的服务。FWD教材的写作方式是按照学习的过程递进的讲述，后续代码要基于前述的代码，因此对于初学者跳跃式的阅读并不是很好的选择。虽然递进的讲述符合学习的规律，但是作者并未给出整体的结构框架，丰富的代码类和函数不断的更新，变量之间关系的复杂性，容易让读者失去方向，因此给出代码工程的统一建模语言（Unified Modeling Language，UML）很必要。

借助代码逆向工程（Reverse Engineering），使用[Visual Paradigm（VP）](https://www.visual-paradigm.com/)⑥完成部分代码的逆向工程实现，目前除了VP，还有[Pyreverse](https://pypi.org/project/pyreverse/)⑦等大量逆向工程工具自动生成UML图表。但是因为代码的复杂性和结构的丰富性，目前很难找到自动生成全部关联的工具，因此下图对于FWD教材案例的UML图表绘制，大部分为手工添加。对Flask结构的把握，主要包括：

A-配置+初始化（主程序）；

B-模板和页面；

C-路由与视图函数；

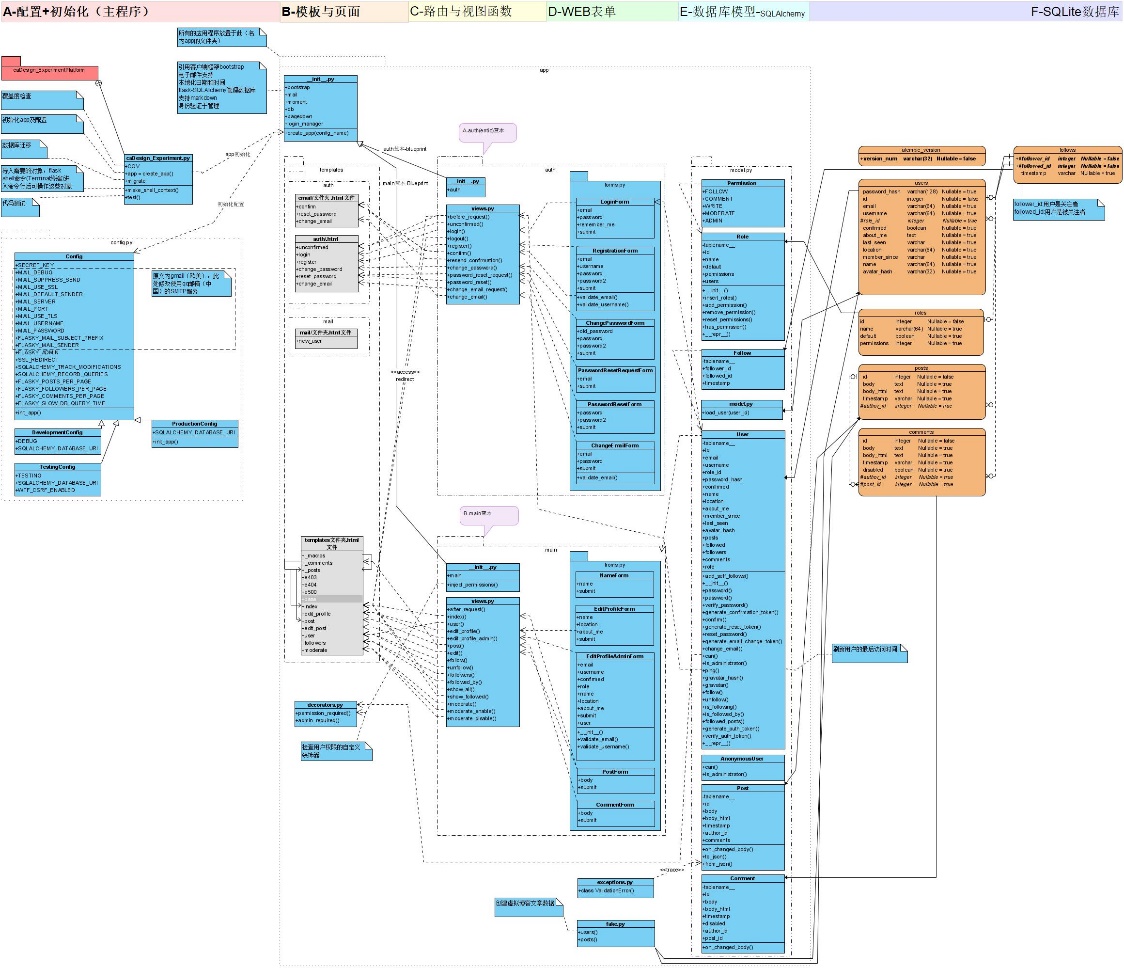
D-Web表单；

E-数据库模型（基于SQLAlchemy）；

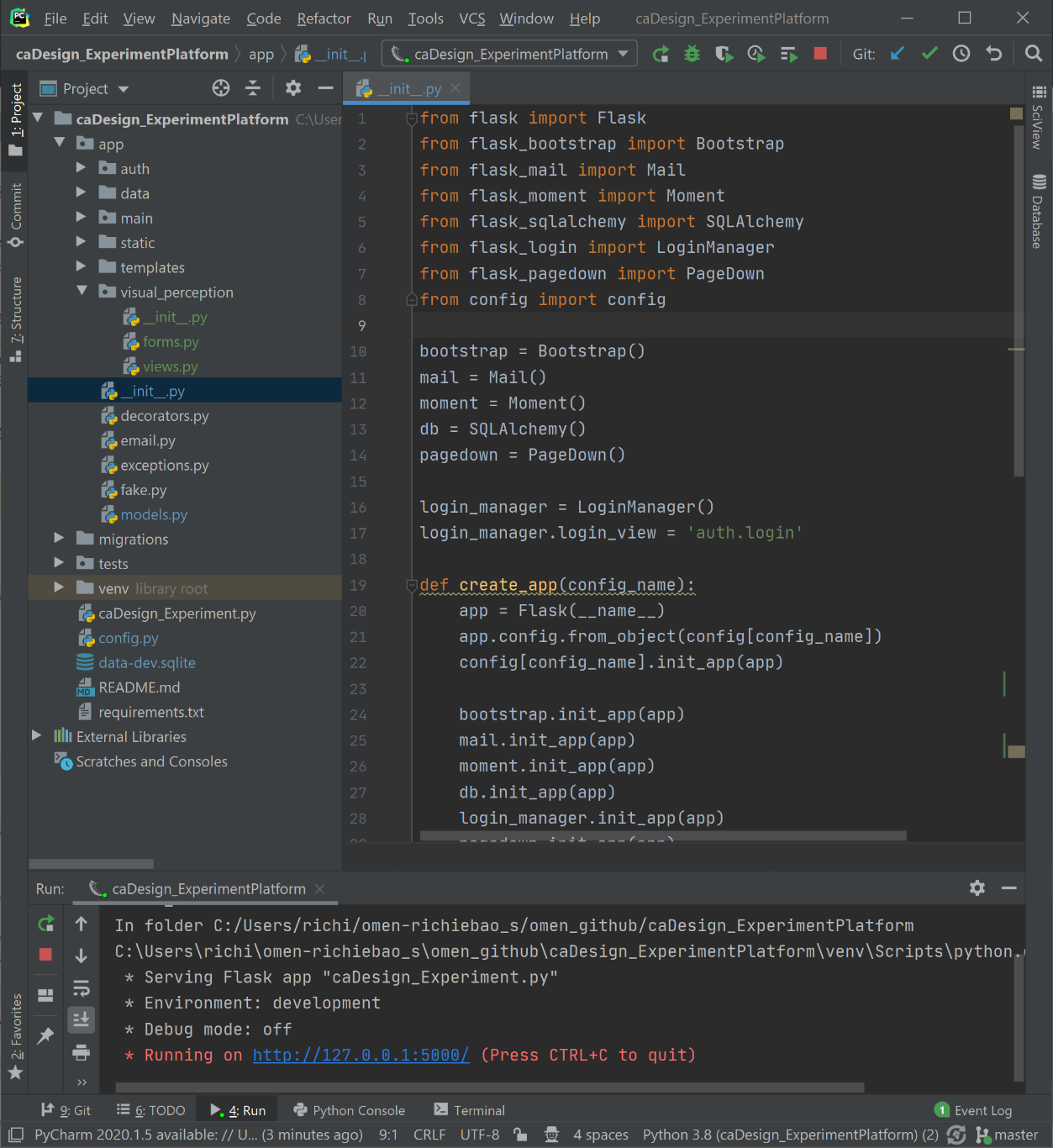
F-数据库（SQLite），等7大部分。

Flask的编写也是抓住这几个部分的关系，完成不同功能实现。因为代码书写的关键是不断的调试来验证已有代码是否顺利运行，达到书写的目的，而Flask是网页的开发，因此对于代码的验证，模板页面部分需要查看页面是否显示正常；数据处理部分仍然可以不断的用print()方法，打印的结果会显示在运行窗口下。建议代码书写前执行set FLASK\_DEBUG=1，这样运行窗口下的错误提示，也将在页面中显示。对于简单的Flask开发，因为实现功能简单，因此实现函数通常位于同一文件之下，但是如果项目工程比较大，众多实现功能如果不加以明确区分，往往造成代码的混乱，因此Flask引入了蓝本（blueprint）的概念，简单讲就是将不同功能实现放置于不同的文件夹（包）下，并构建子文件夹下代码与主程序代码的关联，可以互相调用方法、函数和属性。网络实验部署是采用蓝图的大型应用工厂，使用一个应用进程得到多个应用实例，易于操作。对于Flask大型应用的把握，需要一开始查看文件夹的结构，这反映了当前应用是如何用Flask架构的。

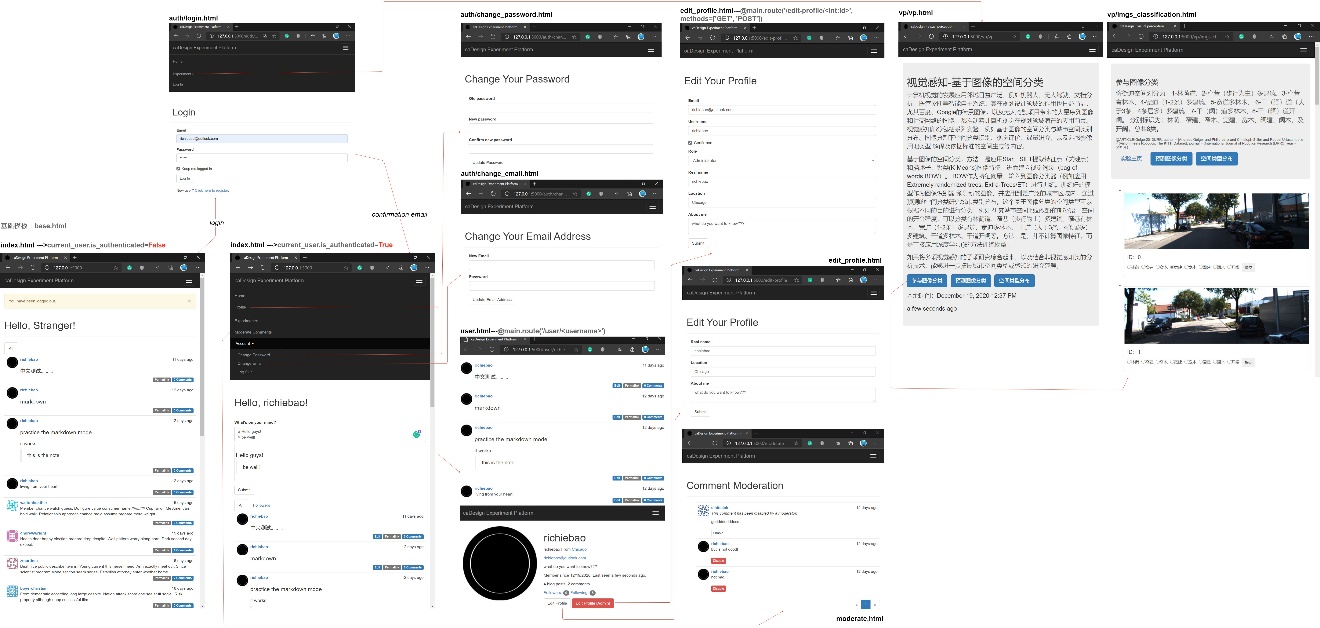
对Flask的理解需要把握应用包（文件夹）的结构，蓝本实现的方法，配置与初始化的关系，路由与视图函数的关系，视图函数与模板页面的关系，Web表单与视图函数的关系，应用数据库模型（表单）读写数据库（SQLite）和与视图函数的关系，以及显示和隐式的代码关系。对隐式代码关系的理解尤为重要，因为突然冒出来的属性变量往往找不到源头，这因为Flask已经帮助完成了相关的任务，只给了输出。同时，因为涉及到页面模板，需要动态的读写视图函数的数据，同时也需要在页面模板内处理数据，Flask应用的[Jinjia](https://jinja.palletsprojects.com/en/2.11.x/)⑧模板引擎实现这些功能，其语法也遵循大部分程序语言的结构。



网络实验平台的工程名为“caDesign\_ExperimentPlatform”（即根目录），配置（config.py）和主程序（caDesign\_Experiment.py），及SQLite数据库（data-dev.sqlite）位于根目录下，migrations为数据库迁移文件夹（可以管理工程版本），test文件夹为测试内容，venv是应用PyCharm建立Flask工程时自动创建的Python环境。所有应用位于app文件包下，static文件夹（系统生成）放置图片,CSS等文件，templates文件夹（系统生成）放置HTML的页面模板，main,auth为FWD教材社交博客的功能应用，data文件夹用于放置相关数据，visual\_perception文件夹为视觉感知-基于图像空间分类实验的网络实现。



下述仅列出了主要的模板页面，类似博客等页面是嵌套在主页等模板页面中，为了便于将其部署于不同模板页面下，通常将此类模板设计成单独的子模板，其模板名为“\_post.html”，方便迁移。此外还有403、404、500等错误页面模板。



### 2.5.2.2 问卷调研

计算机视觉的发展应用领域日益广泛，例如机器人、无人驾驶、文档分析、医疗诊断等智能自主系统，其在规划设计领域的作用也日益凸显， 尤其百度、Google的街景图像，及无人驾驶项目带来的大量序列图像和社交网络的图像，都推动着计算机视觉在规划领域潜在的应用前景。视觉感知部分包括系列实验，例如基于图像的空间分类与城市空间类别分布、图像分割下空间分类识别、视觉评价、绿量研究，及遥感影像用地类型解译和依据标准的空间生成等内容。

基于图像的空间分类，方法一是，应用Star、SIFT提取特征点（关键点）和描述子，聚类(K-Means)图像特征，进而建立视觉词袋（bag-of-words,BOW）。 BOW作为特征向量，输入到图像分类器（例如应用Extremely randomized trees, Extra-Trees/ET）进行训练。训练好的模型作为图像识别器预测新的图像，并应用到更广泛的城市区域内，通过预测的空间分类研究城市类别分布。这个基于图像分类的空间类型可以根据不同的目的进行分类，例如研究城市空间地面视野的郁密度，空间的开合程度，可以分类有林荫道、窄巷（步行为主）多建筑、窄巷有林木、宽道（1-2条）多建筑、宽道多林木、 干道（大于3条，4条居多）多建筑、干道多林木、干道开阔等。方法二是，并不计算图像特征，而是直接应用深度学习的方法训练模型。

如果将多项视觉感知的子项研究综合起来，结合非视觉感知类的分析技术，能够进一步拓展城市空间类型或感知的研究范畴。视觉感知-基于图像的空间分类：问卷调研部分，是使用KITTI数据集中的图像作为城市空间识别的素材，并以FWD教材案例为网络实验平台的基础，在此基础上扩展实验部分内容。其下代码是迁移了指定路径下返回所有文件夹及其下文件结构的代码，列出了’caDesign\_ExperimentPlatform’网络实验平台下app应用文件夹的文件结构，在FWD教材案例基础上增加了文件夹（蓝本）visual\_perception，该阶段包括\_\_init\_\_.py,forms.py和views.py3个文件模块，其中forms.py中基于wtforms库定义了问卷调研表格。因为单选按钮为纵向排列，并没有使用，而是直接在.html模板中直接自行定义。模板文件夹templates下新增了vp文件夹，该阶段包括vp.html,imgs\_classification.html两个文件，分别为”视觉感知-基于图像的空间分类”的说明导航首页，及”参与图像分类”的问卷调研页。

from pathlib import Path  
import util\_misc   
app\_root=r'C:\Users\richi\omen\_richiebao\omen\_github\caDesign\_ExperimentPlatform\app'  
paths=util\_misc.DisplayablePath.make\_tree(Path(app\_root))  
for path in paths:  
 print(path.displayable())

app/  
├── \_\_init\_\_.py  
├── \_\_pycache\_\_/  
│ ├── \_\_init\_\_.cpython-39.pyc  
│ ├── decorators.cpython-39.pyc  
│ ├── email.cpython-39.pyc  
│ ├── exceptions.cpython-39.pyc  
│ ├── fake.cpython-39.pyc  
│ └── models.cpython-39.pyc  
├── auth/  
│ ├── \_\_init\_\_.py  
│ ├── \_\_pycache\_\_/  
│ │ ├── \_\_init\_\_.cpython-39.pyc  
│ │ ├── forms.cpython-39.pyc  
│ │ └── views.cpython-39.pyc  
│ ├── forms.py  
│ └── views.py  
├── data/  
│ └── info\_2011\_09\_26\_drive\_0009\_sync.pkl  
├── decorators.py  
├── email.py  
├── exceptions.py  
├── fake.py  
├── main/  
│ ├── \_\_init\_\_.py  
│ ├── \_\_pycache\_\_/  
│ │ ├── \_\_init\_\_.cpython-39.pyc  
│ │ ├── errors.cpython-39.pyc  
│ │ ├── forms.cpython-39.pyc  
│ │ └── views.cpython-39.pyc  
│ ├── errors.py  
│ ├── forms.py  
│ └── views.py  
├── models.py  
├── static/  
│ ├── favicon.ico  
│ ├── KITTI/  
│ │ ├── 2011\_09\_26\_drive\_0002\_sync/  
│ │ │ ├── 0000000006.png  
│ │ │ ├── 0000000008.png  
│ │ │ ├── 0000000018.png  
│ │ │ ├── 0000000020.png  
│ │ │ ├── 0000000031.png  
│ │ │ ├── 0000000038.png  
│ │ │ ├── 0000000045.png  
│ │ │ └── 0000000069.png  
│ │ ├── 2011\_09\_26\_drive\_0005\_sync/  
│ │ │ ├── 0000000001.png  
│ │ │ ├── 0000000006.png  
 │ └── styles.css  
├── templates/  
│ ├── 403.html  
│ ├── 404.html  
│ ├── 500.html  
│ ├── \_comments.html  
│ ├── \_macros.html  
│ ├── \_posts.html  
│ ├── auth/  
│ │ ├── change\_email.html  
│ │ ├── change\_password.html  
│ │ ├── email/  
│ │ │ ├── change\_email.html  
│ │ │ ├── change\_email.txt  
│ │ │ ├── confirm.html  
│ │ │ ├── confirm.txt  
│ │ │ ├── reset\_password.html  
│ │ │ └── reset\_password.txt  
│ │ ├── login.html  
│ │ ├── register.html  
│ │ ├── reset\_password.html  
│ │ └── unconfirmed.html  
│ ├── base.html  
│ ├── edit\_post.html  
│ ├── edit\_profile.html  
│ ├── followers.html  
│ ├── index.html  
│ ├── mail/  
│ │ ├── new\_user.html  
│ │ └── new\_user.txt  
│ ├── moderate.html  
│ ├── post.html  
│ ├── user.html  
│ └── vp/  
│ ├── img\_prediction.html  
│ ├── imgs\_classification.html  
│ └── vp.html  
├── uploads/  
│ ├── 0000000020.png  
│ ├── 0000000020\_1.png  
│ ├── 0000000030.png  
│ ├── 0000000069.png  
│ ├── 0000000069\_1.png  
│ ├── 01.png  
│ ├── 02.png  
│ ├── 05.png  
│ └── 06.png  
└── visual\_perception/  
 ├── \_\_init\_\_.py  
 ├── \_\_pycache\_\_/  
 │ ├── \_\_init\_\_.cpython-39.pyc  
 │ ├── forms.cpython-39.pyc  
 │ ├── ImageTag\_extractor.cpython-39.pyc  
 │ └── views.cpython-39.pyc  
 ├── forms.py  
 ├── ImageTag\_extractor.py  
 ├── views.py  
 └── vp\_model/  
 ├── ERF\_clf.pkl  
 ├── visual\_BOW.pkl  
 └── visual\_feature.pkl

#### 1）配置视觉感知-基于图像的空间分类蓝本

当建立一个新的子项目时，是在app下建立一个独属的文件夹（蓝本/子包），此处视觉感知实验的蓝本文件夹名为visual\_perception。并在子包中新建\_\_init\_\_.py文件，调入Blueprint，实例化一个蓝本类对象，并调入views(.py)，把路由与蓝本关联起来，即在全局作用域下可以使用该蓝本下的view.py下视图函数及相关方法。同时，需要在app下的\_\_init\_\_.py文件内对应增加配置。

# app/visual\_perception/\_\_init\_\_.py  
from flask import Blueprint  
  
visual\_perception=Blueprint('visual\_perception', \_\_name\_\_) # 参数：蓝本名称，和蓝本所在的包或模块，默认使用\_\_name\_\_。  
from . import views

# app/\_\_init\_\_.py  
from .visual\_perception import visual\_perception as vp\_blueprint  
app.register\_blueprint(vp\_blueprint, url\_prefix='/vp')

#### 2）定义SQLite表结构，写入图像信息

* 定义SQLite表结构

定义两个表结构（模型），*class vp\_imgs(db.Model)*用于存储原始图像的信息，包括图像路径（位于static文件夹下），经纬度和高程信息。*class vp\_classification(db.Model)*表结构，用于存储分类的结果，定义有8个分类，对应c1-c8，同时也引入图像路径字段imgs\_fp，及分类结果classification字段。一开始时，只有vp\_imgs表中有数据，是处理好的图像信息，直接写入到该表中。而vp\_classification表没有数据，只有在模板页面中点击了单选按钮，选择分类提交后，将该信息写入到该表中。同时这两个表之间建立了一对一的关系，指定的外键为db.ForeignKey('vp\_imgs.index')，并建立了双向关系（back\_populates="vp\_imgs"，和back\_populates="vp\_classification"），这样可以直接通过一个表的信息读取另一个表的字段信息。例如在imgs\_classification.html模板中，{{ img\_info.vp\_classification.classification}}的Jinja语句下通过“vp\_imgs”表的一个行信息“img\_info”，链接到“vp\_classification”表中的分类信息“classification”，从而可以在页面中显示当前图像的分类信息。

定义好表结构后，可以在Pycharm的Terminal终端下敲入：flask shell启动shell会话，然后执行from caDesign\_Experiment import db调入数据库实例化对象db，“caDesign\_Experiment”为对应的app名，再执行db.create\_all()，将新建的表结构写入到SQLite数据库（属于库已有的表保持不变）。

# app/models.py  
class vp\_imgs(db.Model):  
 \_\_tablename\_\_ = 'vp\_imgs'  
 index = db.Column(db.Integer, primary\_key=True, autoincrement=True)  
 imgs\_fp=db.Column(db.Text,unique=True,nullable=False)  
 lat=db.Column(db.Float)  
 lon=db.Column(db.Float)  
 alt=db.Column(db.Float)  
 vp\_classification=db.relationship('vp\_classification',uselist=False, back\_populates="vp\_imgs")  
  
class vp\_classification(db.Model):  
 \_\_tablename\_\_ = 'vp\_classification'  
 id=db.Column(db.Integer,primary\_key=True,autoincrement=True)  
 imgs\_fp=db.Column(db.Text,unique=True,nullable=False)  
 c\_1 = db.Column(db.Integer)  
 c\_2 = db.Column(db.Integer)  
 c\_3 = db.Column(db.Integer)  
 c\_4 = db.Column(db.Integer)  
 c\_5 = db.Column(db.Integer)  
 c\_6 = db.Column(db.Integer)  
 c\_7 = db.Column(db.Integer)  
 c\_8 = db.Column(db.Integer)  
 classification=db.Column(db.Text)  
 timestamp=db.Column(db.DateTime,default=datetime.now)  
 index=db.Column(db.Integer,db.ForeignKey('vp\_imgs.index'))  
 vp\_imgs=db.relationship('vp\_imgs',back\_populates="vp\_classification")

* 写入图像信息

先定义表结构，并写入数据库后再向表中写入数据，而不是应用pandas的方法直接默认表结构写入到数据库，这是因为要建立表之间的关系，默认的方式则无法建立表关系。定义函数imgs\_compression\_cv实现图像的压缩，因为图像要在网络页面中显示，较大的图像加载速度慢，影响体验。函数KITTI\_info\_gap()是针对KITTI数据集的操作，因为该数据集是用于无人驾驶场景下计算机视觉算法评测数据集，图像连续，因此通过该函数可以处理KITTI提供的.txt（包含经纬度，高程等信息）文件，保持数据与图像压缩函数给定的参数gap保持一致，即隔一段距离提取一张图像。KITTI\_info2sqlite()函数，则是将提取的图像信息写入到数据库表中，因为表已经存在，因此使用method="append"方法。

def imgs\_compression\_cv(imgs\_root,imwrite\_root,imgsPath\_fp,gap=1,png\_compression=9,jpg\_quality=100):  
 '''  
 function - 使用OpenCV的方法压缩保存图像   
   
 Params:  
 imgs\_root - 待处理的图像文件根目录；string  
 imwrite\_root - 图像保存根目录；string  
 gap - 无人驾驶场景下的图像通常是紧密连续的，可以剔除部分图像避免干扰， 默认值为1；int  
 png\_compression - png格式压缩值，默认为9；int  
 jpg\_quality - jpg格式压缩至，默认为100。for jpeg only. 0 - 100 (higher means better)；int  
 png\_compression: For png only. 0 - 9 (higher means a smaller size and longer compression time):int  
   
 Returns:  
 imgs\_save\_fp - 保存压缩图像文件路径列表；list(string)  
 '''  
 from pathlib import Path  
 import cv2 as cv  
 import numpy as np  
 from tqdm import tqdm  
 import os  
 import pandas as pd   
   
 if not os.path.exists(imwrite\_root):  
 os.makedirs(imwrite\_root)  
   
 imgs\_root=Path(imgs\_root)  
 imgs\_fp=[p for p in imgs\_root.iterdir()][::gap]  
 imgs\_save\_fp=[]  
 for img\_fp in tqdm(imgs\_fp):  
 img\_save\_fp=str(Path(imwrite\_root).joinpath(img\_fp.name))  
 img=cv.imread(str(img\_fp ))  
 if img\_fp.suffix=='.png':  
 cv.imwrite(img\_save\_fp,img,[int(cv.IMWRITE\_PNG\_COMPRESSION), png\_compression])  
 imgs\_save\_fp.append(img\_save\_fp)  
 elif img\_fp.suffix=='.jpg':  
 cv.imwrite(img\_save\_fp,img,[int(cv.IMWRITE\_JPEG\_QUALITY), jpg\_quality])  
 imgs\_save\_fp.append(strimg\_save\_fp)  
 else:  
 print("Only .jpg and .png format files are supported.")  
   
 pd.DataFrame(imgs\_save\_fp,columns=['imgs\_fp']).to\_pickle(imgsPath\_fp)  
 return imgs\_save\_fp  
   
def KITTI\_info\_gap(KITTI\_info\_fp,save\_fp,gap=1):  
 '''  
 function - 读取KITTI文件信息，1-包括经纬度，惯性导航系统信息等的.txt文件。只返回经纬度、海拔信息  
   
 Params:  
 KITTI\_info\_fp - 数据文件路径；string  
 save\_fp - 文件保存路径；string  
 gap - 间隔连续剔除部分图像避免干扰， 默认值为1；int  
   
 Returns:  
 drive\_info\_coordi - 返回经纬度和海拔信息；DataFrame   
   
 '''  
 import pandas as pd  
 from pathlib import Path   
  
 txt\_root=Path(KITTI\_info\_fp)  
 txt\_fp=[str(p) for p in txt\_root.iterdir()][::gap]  
 columns=["lat","lon","alt","roll","pitch","yaw","vn","ve","vf","vl","vu","ax","ay","ay","af","al","au","wx","wy","wz","wf","wl","wu","pos\_accuracy","vel\_accuracy","navstat","numsats","posmode","velmode","orimode"]  
 drive\_info=pd.concat([pd.read\_csv(item,delimiter=' ',header=None) for item in txt\_fp],axis=0)  
 drive\_info.columns=columns  
 drive\_info=drive\_info.reset\_index()   
   
 drive\_info\_coordi=drive\_info[["lat","lon","alt"]]  
 drive\_info\_coordi.to\_pickle(save\_fp)  
  
 return drive\_info\_coordi  
  
def KITTI\_info2sqlite(imgsPath\_fp,info\_fp,replace\_path,db\_fp,table,method='fail'):   
 '''  
 function - 将KITTI图像路径与经纬度信息对应起来，并存入SQLite数据库  
   
 Params:  
 imgsPath\_fp - 图像文件路径；string  
 info\_fp - 图像信息文件路径；string  
 replace\_path - 替换路径名；string  
 db\_fp - SQLite数据库路径；string  
 table - 数据库表名；string  
 method - 包括fail, replace, append等。The default is'fail'；string   
  
 Returns:  
 None  
 '''  
 from pathlib import Path  
 import pandas as pd  
 from sqlalchemy import create\_engine   
   
 imgsPath=pd.read\_pickle(imgsPath\_fp)  
 # flask Jinja的url\_for仅支持'/,因此需要替换'\\'  
 imgsPath\_replace=imgsPath.imgs\_fp.apply(lambda row:str(Path(replace\_path).joinpath(Path(row).name)).replace('\\','/'))  
 info=pd.read\_pickle(info\_fp)  
 imgs\_df=pd.concat([imgsPath\_replace,info],axis=1)   
 engine=create\_engine('sqlite:///'+'\\\\'.join(db\_fp.split('\\')),echo=True)   
  
 try:  
 imgs\_df.to\_sql('%s'%table,con=engine,index=False,if\_exists="%s"%method)  
 print("if\_exists=%s:------Data has been written to the database!"%method)  
 except:  
 print("\_"\*15,'the %s table has been existed...'%table)  
  
  
if \_\_name\_\_=="\_\_main\_\_":  
 imgs\_paths=[r'2011\_09\_26\_drive\_0009\_sync',  
 r'2011\_09\_29\_drive\_0071\_sync',  
 r'2011\_09\_28\_drive\_0001\_sync',  
 r'2011\_09\_29\_drive\_0026\_sync',  
 r'2011\_09\_28\_drive\_0002\_sync',  
 r'2011\_09\_26\_drive\_0117\_sync',  
 r'2011\_09\_26\_drive\_0113\_sync',  
 r'2011\_09\_26\_drive\_0106\_sync',  
 r'2011\_09\_26\_drive\_0104\_sync',  
 r'2011\_09\_26\_drive\_0096\_sync',  
 r'2011\_09\_26\_drive\_0095\_sync',  
 r'2011\_09\_26\_drive\_0093\_sync',  
 r'2011\_09\_26\_drive\_0084\_sync',  
 r'2011\_09\_26\_drive\_0060\_sync',  
 r'2011\_09\_26\_drive\_0059\_sync',  
 r'2011\_09\_26\_drive\_0057\_sync',  
 r'2011\_09\_26\_drive\_0056\_sync',  
 r'2011\_09\_26\_drive\_0051\_sync',  
 r'2011\_09\_26\_drive\_0048\_sync',  
 r'2011\_09\_26\_drive\_0018\_sync',  
 r'2011\_09\_26\_drive\_0017\_sync',  
 r'2011\_09\_26\_drive\_0014\_sync',  
 r'2011\_09\_26\_drive\_0013\_sync',  
 r'2011\_09\_26\_drive\_0011\_sync',  
 r'2011\_09\_26\_drive\_0005\_sync',  
 r'2011\_09\_26\_drive\_0002\_sync',   
 ]  
 g=10  
 i=0  
 for p in imgs\_paths:  
 print("\n%d-%s---处理中..."%(-(len(imgs\_paths[1:])-i),p))  
 imgs\_path=p  
 # A - 使用OpenCV的方法压缩保存图像   
 imgs\_root=r'D:\dataset\KITTI\%s\image\_03\data'%imgs\_path  
 imwrite\_root=r'D:\dataset\KITTI\imgs\_compression\%s'%imgs\_path  
 imgsPath\_fp=r'D:\dataset\KITTI\imgs\_compression\imgsPath\_%s.pkl'%imgs\_path  
 imgs\_save\_fp=imgs\_compression\_cv(imgs\_root,imwrite\_root,imgsPath\_fp,gap=g)  
   
 # B - 读取KITTI经纬度信息，可以指定间隔提取距离  
 KITTI\_info\_fp=r'D:\dataset\KITTI\%s\oxts\data'%imgs\_path  
 save\_fp=r'D:\dataset\KITTI\imgs\_compression\info\_%s.pkl'%imgs\_path  
 drive\_info=KITTI\_info\_gap(KITTI\_info\_fp,save\_fp,gap=g)  
   
 # C - 将文件路径信息写入数据库  
 imgsPath\_fp=r'D:\dataset\KITTI\imgs\_compression\imgsPath\_%s.pkl'%imgs\_path  
 info\_fp=r'D:\dataset\KITTI\imgs\_compression\info\_%s.pkl'%imgs\_path  
 replace\_path=r'KITTI\%s'%imgs\_path  
 db\_fp=r'C:\Users\richi\omen-richiebao\_s\omen\_github\caDesign\_ExperimentPlatform\data-dev.sqlite'  
 KITTI\_info2sqlite(imgsPath\_fp,info\_fp,replace\_path,db\_fp,table='vp\_imgs',method="append")   
   
 i+=1

#### 3）定义路由和视图函数

配置完蓝本之后，在app/visual\_perception/views.py文件下定义视图函数，指定路由（对应的模板页面）。首先可以定义一个简单的视图函数，例如用Flask官方那个最简单的代码来测试是否蓝本配置成功：

@app.route('/hello')  
def hello\_world():  
 return 'Hello, World!'

只是需要重新分配一个路由，例如上述修改为@app.route('/hello')，这样可以在http://127.0.0.1:5000/hello统一资源定位系统（Uniform Resource Locater，URL），即网页地址下打开，如果返回显示“Hello, World!”，则可以说明蓝本配置无误。目前包括两个模板页面vp.html,imgs\_classification.html，分别对应视图函数vp()和imgs\_classification()。vp视图函数网页地址（路由）指向"/vp"，对应模板vp.html下的内容比较简单，只是发布该实验项目的说明，并向模板中传入了current\_time=datetime.datetime.utcnow()参数，可以在模板页面下显示本地时间。imgs\_classification视图函数页面指向”/imgs\_classification”，对应模板页面imgs\_classification.html，因为对图像分类，需要读写数据库，并显示图像，表单提交等动作，要稍显复杂。

首先确定分类的标准，将街道空间划分为：

1. 林荫道；
2. 窄巷（步行为主）多建筑；
3. 窄巷有林木；
4. 宽道（1-2条）多建筑；
5. 宽道多林木；
6. 干（阔）道（大于3条，4条居多）多建筑；
7. 干（阔）道多林木；
8. 干（阔）道开阔。

分别标识为：林荫、窄建、窄木、宽建、宽木、阔建、阔木，及开阔，总共8类。根据分类信息可以在模板中定义表单，定义对应的数据库表结构。视图函数则需要思考“问卷调研”的动作：

1. 首先是读取图像路径信息，传入模板后可以显示图像；
2. 每一图像下对应表单，有8个单选按钮，当选择其中之一后，点击提交按钮，表单信息将返回到视图函数中（POST）；
3. 在视图函数中读取表单信息，将其写入到数据库中（这涉及到数据表结构的设计），不同人点击的分类可能不同，当单击一个分类，则对应写入该分类的数据库表字段中，计数方式为累加。确定图像的最终分类是对应哪个分类的累加数最多；
4. 因为对图像给了分类，分类信息也被写入到对应表中，则可以将分类结果显示在页面中。

#app/visual\_perception/views.py  
from . import visual\_perception  
from flask import render\_template,url\_for, request, session,current\_app,redirect  
import datetime  
from .. import db  
from ..models import vp\_imgs,vp\_classification  
import pandas as pd  
from .forms import imgs\_classi  
  
@visual\_perception.route("/vp",methods=['GET','POST'])  
def vp():  
  
 return render\_template('vp/vp.html',current\_time=datetime.datetime.utcnow())  
  
@visual\_perception.route("/imgs\_classification",methods=['GET','POST'])  
def imgs\_classification():  
 page=request.args.get('page', 1, type=int)  
 query=vp\_imgs.query  
 pagination=query.order\_by(vp\_imgs.index).paginate(page, per\_page=current\_app.config['FLASKY\_POSTS\_PER\_PAGE'],error\_out=False)  
 imgs\_info=pagination.items  
 vp\_classi=vp\_classification.query.all()  
  
 if request.method == 'GET':  
 return render\_template('vp/imgs\_classification.html', imgs\_info=imgs\_info,vp\_classi=vp\_classi,pagination=pagination)  
 else:  
 img\_index=request.form.get('img\_index')  
 img\_fp=request.form.get('img\_fp')  
 classi=int(request.form.get('classi'))  
 img\_current=vp\_classification.query.filter(vp\_classification.imgs\_fp==img\_fp).first()  
 classi\_dic\_value={1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0}  
 classi\_dic\_name={1:u'林荫',2:u'窄建',3:u'窄木',4:u'宽建',5:u'宽木',6:u'阔建',7:u'阔木',8:u'开阔'} # 中文一定要加u，即unicode( str\_name )，否则服务器段如果是py2.7，会提示错误。  
 classi\_dic\_value.update({classi:1})  
 img\_classification=classi\_dic\_name[classi]  
 if not img\_current:  
 img\_classi\_info=vp\_classification(imgs\_fp=img\_fp,  
 c\_1=classi\_dic\_value[1],  
 c\_2=classi\_dic\_value[2],  
 c\_3=classi\_dic\_value[3],  
 c\_4=classi\_dic\_value[4],  
 c\_5=classi\_dic\_value[5],  
 c\_6=classi\_dic\_value[6],  
 c\_7=classi\_dic\_value[7],  
 c\_8=classi\_dic\_value[8],  
 classification=img\_classification,  
 index=img\_index)  
 db.session.add(img\_classi\_info)  
 db.session.commit()  
 else:  
 query\_results=[{1:c\_1,2:c\_2,3:c\_3,4:c\_4,5:c\_5,6:c\_6,7:c\_7,8:c\_8} for c\_1,c\_2,c\_3,c\_4,c\_5,c\_6,c\_7,c\_8 in db.session.query(vp\_classification.c\_1,vp\_classification.c\_2,vp\_classification.c\_3,vp\_classification.c\_4,vp\_classification.c\_5,vp\_classification.c\_6,vp\_classification.c\_7,vp\_classification.c\_8).filter(vp\_classification.imgs\_fp==img\_fp)][0]  
 query\_results\_update=pd.DataFrame.from\_dict(query\_results,orient='index').add(pd.DataFrame.from\_dict(classi\_dic\_value,orient='index'))  
 img\_classification\_=classi\_dic\_name[query\_results\_update.idxmax()[0]]  
 query\_results\_update\_dic=query\_results\_update.squeeze('columns').to\_dict()  
 query\_results\_update\_dic.update({'classification':img\_classification\_})  
 query\_results\_update\_dic\_=dict(zip(['c\_1','c\_2','c\_3','c\_4','c\_5','c\_6','c\_7','c\_8','classification'],query\_results\_update\_dic.values()))  
 vp\_classification.query.filter\_by(imgs\_fp=img\_fp).update(query\_results\_update\_dic\_)  
 db.session.commit()  
  
 return render\_template('vp/imgs\_classification.html',imgs\_info=imgs\_info,vp\_classi=vp\_classi,pagination=pagination) #,form=form

#### 4）定义模板

模板的定义中需要注意对Jinja(2)的使用，[Jinja](https://jinja.palletsprojects.com/en/2.11.x/)是现代而又设计友好，用于Python的模板语言，起源于’Django’。

<!--templates/vp/vp.html -->   
{% extends "base.html" %}  
{% import "bootstrap/wtf.html" as wtf %}  
  
{% block title %}caDesign - visual perception{% endblock %}  
  
{% block page\_content %}  
 <div class="jumbotron">  
 <h1>视觉感知-基于图像的空间分类</h1>  
 <p>  
 计算机视觉的发展应用邻域日益广泛，例如机器人、无人驾驶、文档分析、医疗诊断等智能自主系统。其在规划设计领域的作用也日益凸显，  
 尤其百度、Google的街景图像，以及无人驾驶项目带来的大量序列图像和社交网络的图像，都推动着计算机视觉在规划领域潜在的应用前景。  
 视觉感知部分包括系列实验，例如基于图像的空间分类与城市空间类别分布、图像分割下空间分类识别、视觉评价、绿量研究，以及遥感影像用地类型  
 解译及依据标准的空间生成等内容。  
 </p>  
 <p>  
 基于图像的空间分类，方法一是应用Star、SIFT提取特征点（关键点）和描述子，聚类(K-Means)图像特征，进而建立视觉词袋（bag-of-words,BOW）。  
 BOW作为特征向量，输入到图像分类器（例如应用Extremely randomized trees, Extra-Trees/ET）进行训练。训练好的模型作为图像识别器  
 预测新的图像，并应用到更广泛的城市区域内，通过预测的空间分类研究城市类别分布。这个基于图像分类的空间类型可以根据不同的目的进行分类，例如  
 研究城市空间地面视野的郁密度，空间的开合程度，可以分类有林荫道、窄巷（步行为主）多建筑、窄巷有林木、宽道（1-2条）多建筑、宽道多林木、  
 干道（大于3条，4条居多）多建筑、干道多林木、干道开阔等。方法二是，并不计算图像特征，而是直接应用深度学习()的方法训练模型。  
 </p>  
 <p>  
 如果将多项视觉感知的子项研究综合起来，以及结合非视觉感知类的分析技术，能够进一步拓展城市空间类型或感知的研究范畴。  
 </p>  
  
 <p>  
 <a class="btn btn-primary btn-lg" href="{{ url\_for('visual\_perception.imgs\_classification') }}" role="button">参与图像分类</a>  
 &nbsp<a class="btn btn-primary btn-lg" href="{{ url\_for('visual\_perception.vp') }}" role="button">预测图像分类</a>  
 &nbsp<a class="btn btn-primary btn-lg" href="{{ url\_for('visual\_perception.vp') }}" role="button">空间类型分布</a>  
 </p>  
 <p>本地时间：{{ moment(current\_time).format('LLL') }}</p>  
 <p>{{ moment(current\_time).fromNow(refresh=True) }}</p>  
 </div>  
{% endblock %}

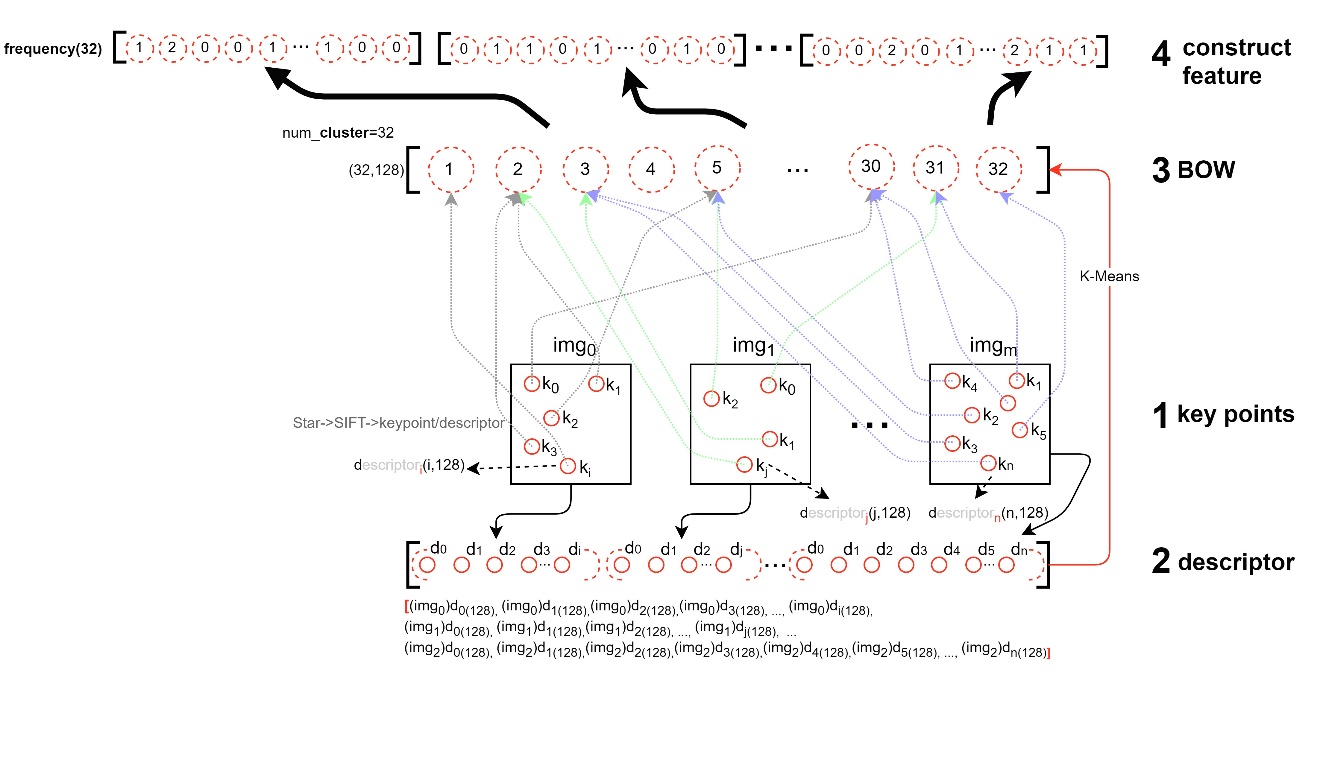
<!--templates/vp/imgs\_classification.html -->   
{% extends "base.html" %}  
{% import "bootstrap/wtf.html" as wtf %}  
{% import "\_macros.html" as macros %}  
  
{% block title %}caDesign - visual perception{% endblock %}  
  
{% block page\_content %}  
  
 <div class="jumbotron">  
 <h3>参与图像分类</h3>  
 <p>  
 将街道空间划分为：1-林荫道、2-窄巷（步行为主）多建筑、3-窄巷有林木、4-宽道（1-2条）多建筑、5-宽道多林木、 6-干（阔）道（大于3条，4条居多）多建筑、7-干（阔）道多林木、8-干（阔）道开阔。  
 分别标识为：林荫、窄建、窄木、宽建、宽木、阔建、阔木，及开阔，总共8类。  
 </p>  
  
 <h6>@ARTICLE{Geiger2013IJRR, author = {Andreas Geiger and Philip Lenz and Christoph Stiller and Raquel Urtasun}, title = {Vision meets Robotics: The KITTI Dataset}, journal = {International Journal of Robotics Research (IJRR)}, year = {2013} }</h6>  
  
 <p>  
 <a class="btn btn-secondary btn-lg" href="{{ url\_for('visual\_perception.vp') }}" role="button">实验主页</a>  
 &nbsp<a class="btn btn-primary btn-lg" href="{{ url\_for('visual\_perception.vp') }}" role="button">预测图像分类</a>  
 &nbsp<a class="btn btn-primary btn-lg" href="{{ url\_for('visual\_perception.vp') }}" role="button">空间类型分布</a>  
 </p>  
 </div>  
  
 <ul class="question-list-group">  
 {% for img\_info in imgs\_info %}  
 <li style="float:left">  
 <div class="row">  
 <div class=" col-md-10">  
 <div class="thumbnail">  
 <img src="{{ url\_for('static',filename=img\_info.imgs\_fp[7:]) }}" alt="">  
 <div class="caption">  
 <h4>ID：{{ img\_info.index }}  
 {{ img\_info.vp\_classification.classification}}  
 </h4>  
 <iframe name="formDestination" class="iframe", style="display:none;"></iframe>  
 <form action="" method="post" target="formDestination">  
 <input type="radio" name="classi" value="1"/>林荫&nbsp  
 <input type="radio" name="classi" value="2"/>窄建&nbsp  
 <input type="radio" name="classi" value="3"/>窄木&nbsp  
 <input type="radio" name="classi" value="4"/>宽建&nbsp  
 <input type="radio" name="classi" value="5"/>宽木&nbsp  
 <input type="radio" name="classi" value="6"/>阔建&nbsp  
 <input type="radio" name="classi" value="7"/>阔木&nbsp  
 <input type="radio" name="classi" value="8"/>开阔&nbsp  
  
 <input type="hidden" name="img\_index" value="{{img\_info.index }}">  
 <input type="hidden" name="img\_fp" value="{{ img\_info.imgs\_fp}}">  
  
 <input type="submit" value="提交" class="btn btn-secondary btn-sm" onClick="this.form.submit(); this.disabled=true; this.value='已提交'; ">  
 </form>  
 </div>  
 </div>  
 </div>  
  
 </div>  
 </li>  
 {% endfor %}  
 </ul>  
  
{% if pagination %}  
<div class="pagination">  
 {{ macros.pagination\_widget(pagination, 'visual\_perception.imgs\_classification') }}  
</div>  
{% endif %}  
  
{% endblock %}



### 2.5.2.3 视觉词袋与构建图像映射特征

计算每一张图像的关键点描述子（一个关键点描述子有128维，一幅图像有多个关键点），把所有图像的关键点描述子集合在一起，就是所有图像特征的集合。因为每一关键点描述子均不同，因此使用聚类的方法指定聚类的数量（32）聚合特征，即聚合所有图像的关键描述子为指定数量构造码本（Bag of Words，BOW），并建立聚类模型。该码本就是所有图像特征（聚类后）的集合，因为每一图像的关键点描述子都可以在该码本中找到对应的特征（32个聚类的1个或多个聚类组合），因此可以用码本编码每一图像（用聚类模型预测）。保持码本的形状（shape，32），将一幅图像的所有关键点描述子对应到码本的编号上，因为聚类的结果，一个编号通常对应有多个关键点，计算所有对应到编号上关键点的频数（有的编码频数也会为0，即没有对应的关键点），这个对应码本编号频数的结果就反映了该图像特征。

码本是将所有可用的码（聚类）放在一起，组成类似字典的表，用序号给不同的码编号（或者是列表的排序）。如果要对一个单词（或一句话）编码，则可以应用码本编号。



读取“问卷调研”的数据库结果。因为数据库中存储的图像路径为相对路径，而此时的代码可能位于他处，因此将其转换为绝对路径。

import sqlite3  
import pandas as pd  
import util\_misc  
import os  
  
db\_fp=r'C:\Users\richi\omen\_richiebao\omen\_github\caDesign\_ExperimentPlatform\data-dev.sqlite'  
vp\_classification=pd.read\_sql\_table('vp\_classification', 'sqlite:///%s'%db\_fp) # pd.read\_sql\_table从数据库中读取指定的表  
vp\_classification['absolute\_imags\_fp']=vp\_classification.apply(lambda row:os.path.join(r'C:\Users\richi\omen-richiebao\_s\omen\_github\caDesign\_ExperimentPlatform\app\static',row.imgs\_fp),axis=1)  
util\_misc.print\_html(vp\_classification)

|  | **id** | **imgs\_fp** | **c\_1** | **c\_2** | **c\_3** | **c\_4** | **c\_5** | **c\_6** | **c\_7** | **c\_8** | **classification** | **timestamp** | **index** | **absolute\_imags\_fp** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | KITTI/2011\_09\_26\_drive\_0009\_sync/0000000000.png | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 宽木 | 2020-12-20 13:10:36.725045 | 1 | C:\Users\richi\omen-richiebao\_s\omen\_github\caDesign\_ExperimentPlatform\app\static\KITTI/2011\_09\_26\_drive\_0009\_sync/0000000000.png |
| **1** | 2 | KITTI/2011\_09\_26\_drive\_0009\_sync/0000000010.png | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 宽木 | 2020-12-20 13:10:42.138661 | 2 | C:\Users\richi\omen-richiebao\_s\omen\_github\caDesign\_ExperimentPlatform\app\static\KITTI/2011\_09\_26\_drive\_0009\_sync/0000000010.png |
| **2** | 3 | KITTI/2011\_09\_26\_drive\_0009\_sync/0000000020.png | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 宽木 | 2020-12-20 13:10:46.728137 | 3 | C:\Users\richi\omen-richiebao\_s\omen\_github\caDesign\_ExperimentPlatform\app\static\KITTI/2011\_09\_26\_drive\_0009\_sync/0000000020.png |
| **3** | 4 | KITTI/2011\_09\_26\_drive\_0009\_sync/0000000030.png | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 宽木 | 2020-12-20 13:10:55.079670 | 4 | C:\Users\richi\omen-richiebao\_s\omen\_github\caDesign\_ExperimentPlatform\app\static\KITTI/2011\_09\_26\_drive\_0009\_sync/0000000030.png |
| **4** | 5 | KITTI/2011\_09\_26\_drive\_0009\_sync/0000000040.png | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 宽木 | 2020-12-20 13:10:58.771853 | 5 | C:\Users\richi\omen-richiebao\_s\omen\_github\caDesign\_ExperimentPlatform\app\static\KITTI/2011\_09\_26\_drive\_0009\_sync/0000000040.png |

print("分类频数统计：\n",vp\_classification.classification.value\_counts())

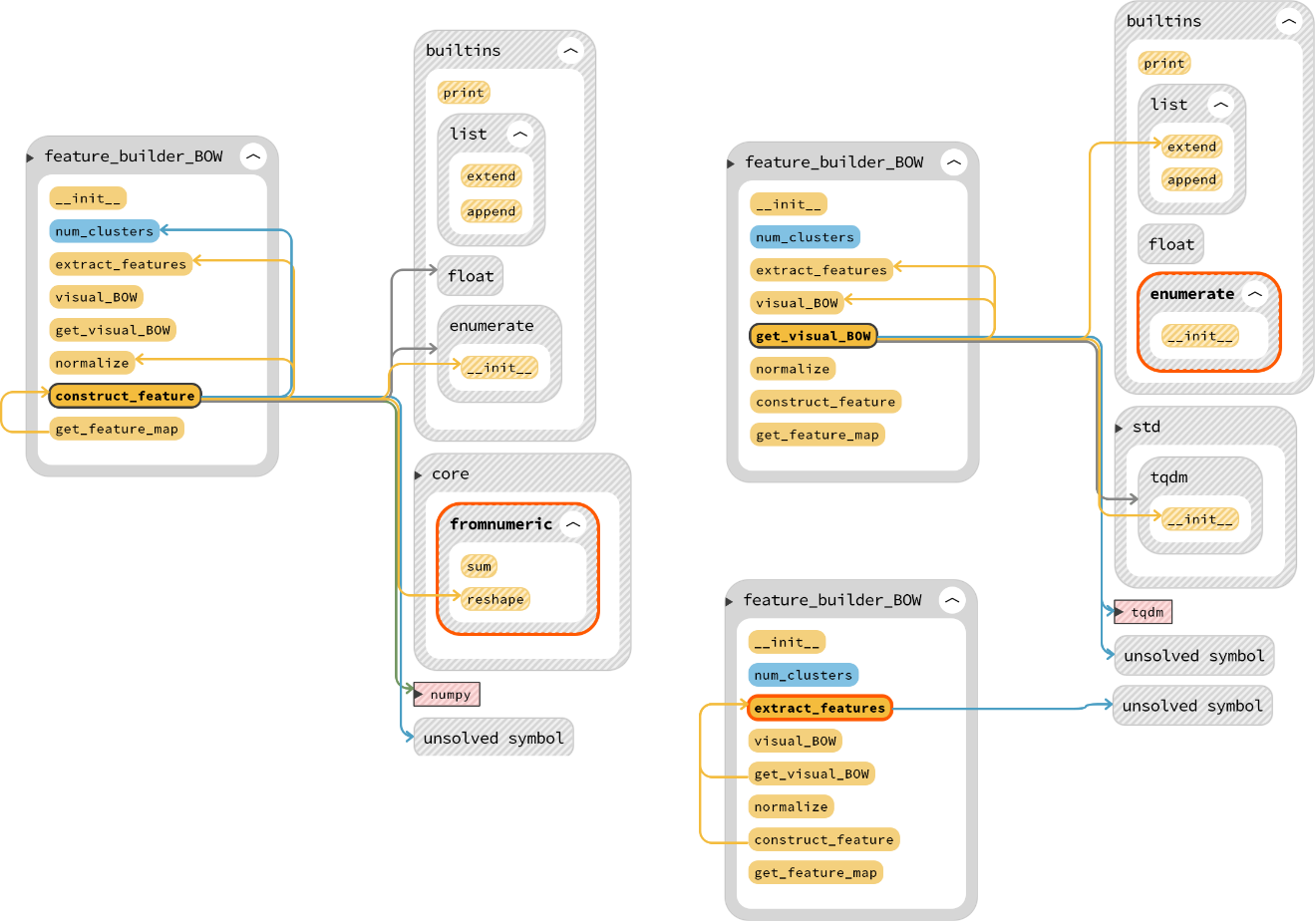
分类频数统计：  
 阔木 189  
窄建 188  
宽木 104  
开阔 85  
窄木 78  
宽建 60  
阔建 47  
林荫 47  
Name: classification, dtype: int64

以一张图像的分类结果为键，以图像绝对路径为名建立对应到一张图像的字典。然后将所有图像的字典放置于一个列表中，用于构造所有图像的码本和计算每一图像的特征映射。

def load\_training\_data(imgs\_df,classi\_field,classi\_list,path\_field):   
 '''  
 function - 按照分类提取图像路径与规律  
   
 Params:  
 imgs\_df - 由Pandas读取的数据库表数据，含分类信息；DataFrame  
 classi\_field - 分类字段；string  
 classi\_list - 分类列表；list(string)  
 path\_field - 图像绝对路径名；string  
   
 Returns:  
 训练数据；list(dict)  
 '''  
 import pandas as pd  
   
 imgs\_group=imgs\_df.groupby(['classification'])  
 training\_data=[[{'object\_class':classi,'image\_path':row} for row in imgs\_group.get\_group(classi)[path\_field].tolist()] for classi in classi\_list]  
 flatten\_lst=lambda lst: [m for n\_lst in lst for m in flatten\_lst(n\_lst)] if type(lst) is list else [lst]  
 return flatten\_lst(training\_data)  
  
classi\_list=['林荫','窄建','窄木','宽建','宽木','阔建','阔木','开阔']   
classi\_field='classification'   
path\_field='absolute\_imags\_fp'  
training\_data=load\_training\_data(vp\_classification,classi\_field,classi\_list,path\_field)   
print(training\_data[:5])

[{'object\_class': '林荫', 'image\_path': 'C:\\Users\\richi\\omen-richiebao\_s\\omen\_github\\caDesign\_ExperimentPlatform\\app\\static\\KITTI/2011\_09\_26\_drive\_0117\_sync/0000000657.png'}, {'object\_class': '林荫', 'image\_path': 'C:\\Users\\richi\\omen-richiebao\_s\\omen\_github\\caDesign\_ExperimentPlatform\\app\\static\\KITTI/2011\_09\_26\_drive\_0117\_sync/0000000036.png'}, {'object\_class': '林荫', 'image\_path': 'C:\\Users\\richi\\omen-richiebao\_s\\omen\_github\\caDesign\_ExperimentPlatform\\app\\static\\KITTI/2011\_09\_26\_drive\_0117\_sync/0000000354.png'}, {'object\_class': '林荫', 'image\_path': 'C:\\Users\\richi\\omen-richiebao\_s\\omen\_github\\caDesign\_ExperimentPlatform\\app\\static\\KITTI/2011\_09\_26\_drive\_0117\_sync/0000000038.png'}, {'object\_class': '林荫', 'image\_path': 'C:\\Users\\richi\\omen-richiebao\_s\\omen\_github\\caDesign\_ExperimentPlatform\\app\\static\\KITTI/2011\_09\_26\_drive\_0117\_sync/0000000196.png'}]

如果代码量比较大，逐行的分析代码之间互相调用的关系是费时和费力的事情，因此借助逆向工程。代码逆向工程的工具很多，这里使用了[Sourcetrail](https://github.com/CoatiSoftware/Sourcetrail)⑨工具。定义feature\_builder\_BOW构造视觉码本和提取图像映射特征的类之后，主要调用了两个方法，先用feature\_builder\_BOW().get\_visual\_BOW(training\_data,)构造码本，再用 feature\_builder\_BOW().get\_feature\_map(training\_data,kmeans)返回每一图像映射特征。从下述逆向工程分析结果能够很清楚的梳理出两次类方法的调用所关联调用的其它类中函数。.get\_visual\_BOW函数内调用有extract\_features和visual\_BOW两个函数，同时显示了调用的内嵌方法，print，enumerate，tqdm等；.get\_feature\_map则调用有extract\_features和normalize两个函数，同时使用了类变量self.num\_clusters。也可以查看extract\_features分别被.get\_visual\_BOW和.get\_feature\_map调用。



class feature\_builder\_BOW:  
 '''  
 class - 根据所有图像关键点描述子聚类建立图像视觉词袋，获取每一图像的特征（码本）映射的频数统计  
 '''   
 def \_\_init\_\_(self,num\_cluster=32):  
 self.num\_clusters=num\_cluster  
  
 def extract\_features(self,img):   
 '''  
 function - 提取图像特征  
   
 Params:  
 img - 读取的图像  
 '''  
 import cv2 as cv  
   
 star\_detector=cv.xfeatures2d.StarDetector\_create()  
 key\_points=star\_detector.detect(img)  
 img\_gray=cv.cvtColor(img,cv.COLOR\_BGR2GRAY)  
 kp,des=cv.xfeatures2d.SIFT\_create().compute(img\_gray, key\_points) # SIFT特征提取器提取特征  
 return des  
   
 def visual\_BOW(self,des\_all):   
 '''  
 function - 聚类所有图像的特征（描述子/SIFT），建立视觉词袋  
   
 Params:  
 des\_all - 所有图像的关键点描述子  
 '''  
 from sklearn.cluster import KMeans  
   
 print("start KMean...")  
 kmeans=KMeans(self.num\_clusters)  
 kmeans=kmeans.fit(des\_all)  
 # centroids=kmeans.cluster\_centers\_  
 print("end KMean...")  
 return kmeans   
   
 def get\_visual\_BOW(self,training\_data):  
 '''  
 function - 提取图像特征，返回所有图像关键点聚类视觉词袋  
   
 Params:  
 training\_data - 训练数据集  
 '''  
 import cv2 as cv  
 from tqdm import tqdm  
   
 des\_all=[]   
 for item in tqdm(training\_data):   
 classi\_judge=item['object\_class']  
 img=cv.imread(item['image\_path'])  
 des=self.extract\_features(img)  
 des\_all.extend(des)   
 kmeans=self.visual\_BOW(des\_all)   
 return kmeans  
   
 def normalize(self,input\_data):   
 '''  
 fuction - 归一化数据  
   
 Params:  
 input\_data - 待归一化的数组  
 '''  
 import numpy as np  
   
 sum\_input=np.sum(input\_data)  
 if sum\_input>0:  
 return input\_data/sum\_input # 单一数值/总体数值之和，最终数值范围[0,1]  
 else:  
 return input\_data   
   
 def construct\_feature(self,img,kmeans):   
 '''  
 function - 使用聚类的视觉词袋构建图像特征（构造码本）  
   
 Paras:  
 img - 读取的单张图像  
 kmeans - 已训练的聚类模型  
 '''  
 import numpy as np  
   
 des=self.extract\_features(img)  
 labels=kmeans.predict(des.astype(float)) # 对特征执行聚类预测类标  
 feature\_vector=np.zeros(self.num\_clusters)  
 for i,item in enumerate(feature\_vector): # 计算特征聚类出现的频数/直方图  
 feature\_vector[labels[i]]+=1  
 feature\_vector\_=np.reshape(feature\_vector,((1,feature\_vector.shape[0])))  
 return self.normalize(feature\_vector\_)  
   
 def get\_feature\_map(self,training\_data,kmeans):   
 '''  
 function - 返回每个图像的特征映射（码本映射）  
   
 Paras:  
 training\_data - 训练数据集  
 kmeans - 已训练的聚类模型  
 '''  
 import cv2 as cv  
   
 feature\_map=[]  
 for item in training\_data:  
 temp\_dict={}  
 temp\_dict['object\_class']=item['object\_class']  
 img=cv.imread(item['image\_path'])  
 temp\_dict['feature\_vector']=self.construct\_feature(img,kmeans)  
 if temp\_dict['feature\_vector'] is not None:  
 feature\_map.append(temp\_dict)  
  
 return feature\_map  
   
import util\_misc  
s\_t=util\_misc.start\_time()  
  
# 修改发生了改变的路径  
training\_data\_=[]  
for i in training\_data:  
 path\_update=i['image\_path'].replace(r"C:\Users\richi\omen-richiebao\_s",r"C:\Users\richi\omen\_richiebao")  
 i.update({'image\_path':path\_update})  
 training\_data\_.append(i)  
  
kmeans=feature\_builder\_BOW().get\_visual\_BOW(training\_data\_,)   
print("\_"\*50)  
  
import pickle  
with open('./model/visual\_BOW.pkl','wb') as f: # 使用with结构避免手动的文件关闭操作  
 pickle.dump(kmeans,f) # 存储kmeans聚类模型  
   
feature\_map=feature\_builder\_BOW().get\_feature\_map(training\_data\_,kmeans)   
with open('./model/visual\_feature.pkl','wb') as f:  
 pickle.dump(feature\_map,f) # 存储图像特征  
  
util\_misc.duration(s\_t)

start time: 2022-01-08 18:01:30.911273  
  
  
100%|██████████| 798/798 [00:57<00:00, 13.91it/s]  
  
  
start KMean...  
end KMean...  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
end time: 2022-01-08 18:08:38.278630  
Total time spend:7.12 minutes

### 2.5.2.4 决策树（Decision trees）与随机森林（Random forests）

#### 1）决策树

*Mastering Machine Learning with scikit-learn*[2] 对决策树和随机森林算法有清晰易于理解的阐述，这里以其为蓝本，应用案例数据，辅助Python计算和应用sklearn库加以说明。首先，录入猫狗分类的特征数据集catDog\_trainingData\_df，所录入的数据为文本类型，将其通过pandas库提供的pd.get\_dummies方法转换为独热编码（One-Hot Encoding），主要应用于数据集的特征列；同时也应用了sklearn库的preprocessing.LabelEncoder()将其转换为整数编码，主要用于数据集的类标列。

决策树（见下文中定义函数decisionTree\_structure()计算的流程图）测试特征的内部节点，用盒子表示，节点之间通过边来连接，边表示了测试的可能输出，（根据阈值）将训练实例分到不同的子集中。子节点应用特征值继续测试训练实例的子集，直到满足一个停止标准。分类任务中，决策树中不在分支的节点为叶节点，表示类别（如果是在回归任务中，一个叶节点包含多个实例，这些实例对应的响应变量值可以通过求均值来估计这个叶节点对应的响应变量）；带有分支的节点通常称为分支节点（或子节点）。在决策树构建完成后，对于一个测试实例进行预测，只需要从根节点顺着对应的边到达某个叶节点。

训练决策树的优化算法，使用Ross Quinlan发明的迭代二叉树3代（Iterative Dichotomiser 3，ID3）的算法。

import pandas as pd  
catDog\_trainingData\_df=pd.DataFrame({'plays\_Fetch':['Yes','No','No','No','No','No','No','No','No','Yes','Yes','No','Yes','Yes'],  
 'is\_grumpy':['No','Yes','Yes','Yes','No','Yes','Yes','No','Yes','No','No','No','Yes','Yes'],  
 'fvorite\_food':['bacon','dog\_food','cat\_food','bacon','cat\_food','bacon','cat\_food','dog\_food','cat\_food','dog\_food','bacon','cat\_food','cat\_food','bacon'],  
 'species':['dog','dog','cat','cat','cat','cat','cat','dog','cat','dog','dog','cat','cat','dog']})  
print("原始数据：\n",catDog\_trainingData\_df)

原始数据：  
 plays\_Fetch is\_grumpy fvorite\_food species  
0 Yes No bacon dog  
1 No Yes dog\_food dog  
2 No Yes cat\_food cat  
3 No Yes bacon cat  
4 No No cat\_food cat  
5 No Yes bacon cat  
6 No Yes cat\_food cat  
7 No No dog\_food dog  
8 No Yes cat\_food cat  
9 Yes No dog\_food dog  
10 Yes No bacon dog  
11 No No cat\_food cat  
12 Yes Yes cat\_food cat  
13 Yes Yes bacon dog

数据集的特征值完成独热编码后，或增加相应的列，对于仅存在有两个分类的列（例如“Yes”和“No”），虽然增加了新的一列，但是与单独一列实质上并没有区别（非1即0），因此在决策树中使用二者中的任何一列都是一样的（例如play\_No或play\_Yes）。但是分类在3类及其以上者，当用独热编码完成转换后，新增列之间是不能互相替换，例如food\_bacon、food\_cat food和food\_dog food。

def df\_multiColumns\_LabelEncoder(df,columns=None):   
 '''  
 function - 根据指定的（多个）列，将分类转换为整数表示，区间为[0,分类数-1]  
   
 Params:  
 df - DataFrame格式数据；DataFrame  
 columns - 指定待转换的列名列表；list(string)  
   
 Returns:  
 output - 分类整数编码；DataFrame  
 '''  
 from sklearn import preprocessing  
   
 output=df.copy()  
 if columns is not None:  
 for col in columns:  
 output[col]=preprocessing.LabelEncoder().fit\_transform(output[col])  
 else:  
 for column\_name, col in output.iteritems():  
 output[column\_name]=preprocessing.LabelEncoder().fit\_transform(col)  
   
 return output   
   
catDog\_trainingData\_df\_encoder=df\_multiColumns\_LabelEncoder(df=catDog\_trainingData\_df,columns=['plays\_Fetch','is\_grumpy','fvorite\_food','species'])   
print("encoder of each column\n",catDog\_trainingData\_df\_encoder)  
  
catDog\_trainingData\_dummies=pd.get\_dummies(catDog\_trainingData\_df,prefix=['play','grumpy','food','species'])  
print("fequency of each column:\n",catDog\_trainingData\_dummies.apply(pd.Series.value\_counts))  
print('one-hot(dummies):\n')  
import util\_misc  
util\_misc.print\_html(catDog\_trainingData\_dummies,row\_numbers=14)

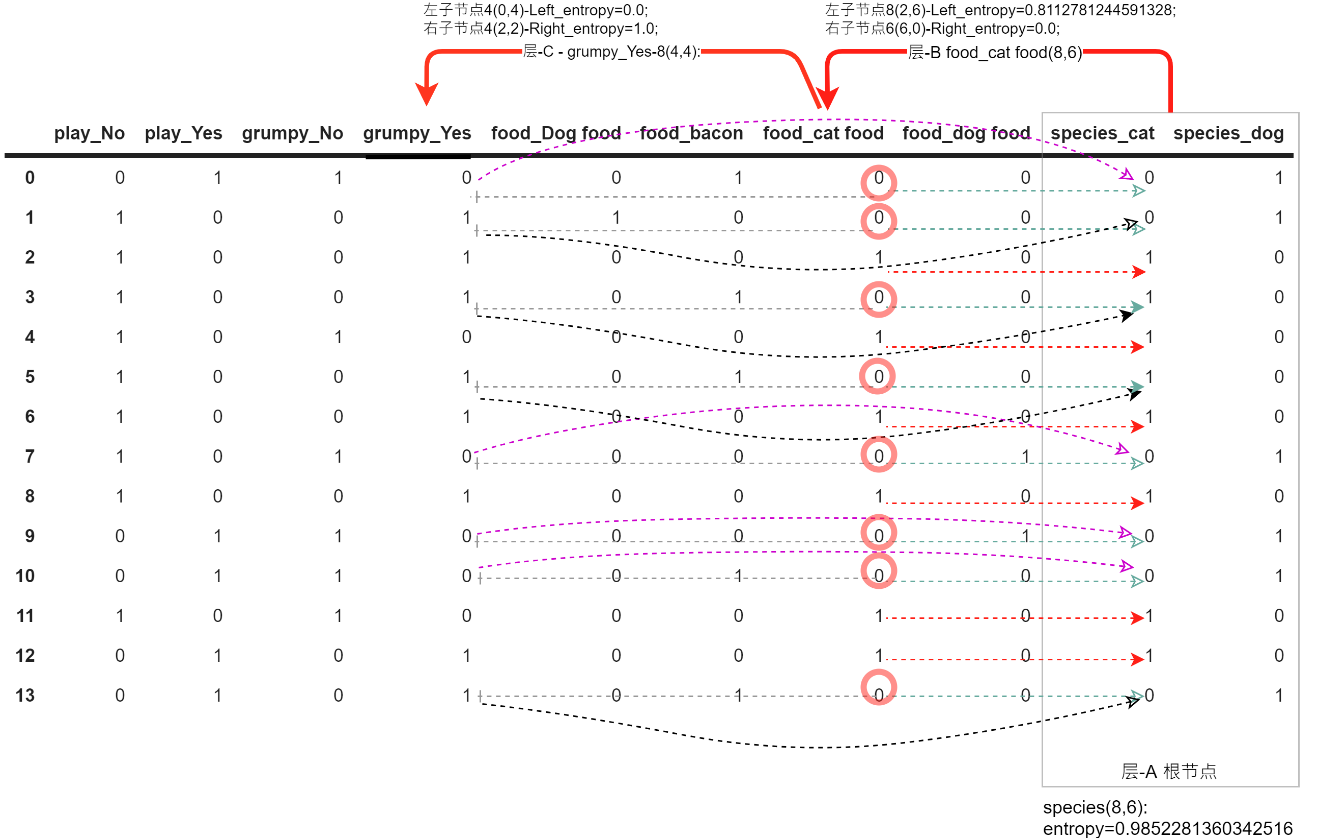
encoder of each column  
 plays\_Fetch is\_grumpy fvorite\_food species  
0 1 0 0 1  
1 0 1 2 1  
2 0 1 1 0  
3 0 1 0 0  
4 0 0 1 0  
5 0 1 0 0  
6 0 1 1 0  
7 0 0 2 1  
8 0 1 1 0  
9 1 0 2 1  
10 1 0 0 1  
11 0 0 1 0  
12 1 1 1 0  
13 1 1 0 1  
fequency of each column:  
 play\_No play\_Yes grumpy\_No grumpy\_Yes food\_bacon food\_cat\_food \  
0 5 9 8 6 9 8   
1 9 5 6 8 5 6   
  
 food\_dog\_food species\_cat species\_dog   
0 11 6 8   
1 3 8 6   
one-hot(dummies):

|  | **play\_No** | **play\_Yes** | **grumpy\_No** | **grumpy\_Yes** | **food\_bacon** | **food\_cat\_food** | **food\_dog\_food** | **species\_cat** | **species\_dog** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| **1** | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| **2** | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **3** | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| **4** | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| **5** | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| **6** | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **7** | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| **8** | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **9** | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| **10** | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| **11** | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| **12** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **13** | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |

决策树通过检测一个特征序列的值来估计响应变量的值。即能产出只包含猫和只包含狗的子集的检测，要优于一个产出中同时包含猫狗的检测。因为一个子集中的成员同时包含不同的类，无法确定实例的分类。对于这个检测可以使用熵的衡量方式量化不确定性的程度（单位为比特，bits）。其公式为：，式中 是输出的概率，常见值为2、e和10。由于一个小于1的数值的对数为负数，求和为负数，因此取反。为了方便计算一个对象的熵来查看计算的流程，定义entropy\_compomnent()函数实现。

找出对分类动物最有帮助的特征，即找出能把熵降到最低的特征（熵值越大，类别分布越均匀；熵值越小，类别分布越集中，即分布不均匀）。下述代码的过程是按照下图的执行指向过程计算，在层-A根节点中，根据类标计算猫（8只）狗（6只）熵值为0.985。先利用food\_cat food特征列，左子节点不吃猫食（0）对应的狗分类为6只，猫分类为2只，信息熵为0.811；右子节点吃猫食（1）对应的狗分类为0只，猫分类为6只，信息熵为0.0；因为左子节点信息熵0.811>0.5，因此需要对左子节点继续检测。选择grumpy\_Yes特征列对应左子节点，即food\_cat food中值为0（不吃猫食）的行（实例），包括6只狗和2只猫。这8个实例可以根据grumpy\_Yes特征列，即是否脾气暴躁（grumpy）划分为两类，其中脾气暴躁（1）的为4个实例（右子节点），而脾气不暴燥（0）也为4个实例（左子节点）。将其分布对应到类标列，可以得知脾气暴躁的4个实例中，为猫分类的有2个，为狗分类的亦有2个，信息熵为1；而脾气不暴躁的4个实例中，为猫分类的为0，为狗的分类为4，信息熵为0。就是说明，如果不吃猫食，而脾气不暴躁的实例分类为狗。

上述的决策树，根据两个特征列，food\_cat food和grumpy\_Yes判断物种分类（类标），因为层-C存在熵值为1的右子节点，需要继续删选特征，继续检测。例如想判断第12行实例（样本）的类标，已知food\_cat food的值为1（吃猫食），grumpy\_Yes的值为1（脾气暴躁），通过层-B可以判断出，该实例会被分配到右子节点（吃猫食），该层右子节点信息熵为0，因此可以判断，该实例为猫，与实际相符。再例如，对于第10行实例，已知’food\_cat food的值为0（不吃猫食），grumpy\_Yes的值为0（脾气不暴躁），通过层-B可以判断出，该实例会被分配到左子节点（不吃猫食），因为该节点信息熵大于0.5，因此继续用层-C检测，由脾气不暴躁，将其指向左子节点，该节点信息熵为0，因此可以判定该实例为狗，与实际相符。



* 信息增益（Information gain）

上述的计算直接选择了两个特征列，实际上是需要判断选择哪些特征列用于检测，以减少分类的不确定性。在层-B中，产生了两个子集（子节点），熵分别为0.811和0.0，其平均熵为，而根节点的熵为0.985，最大不确定性的熵为1。衡量熵的减少可以使用信息增益的指标，其公式为： xaaa*υ*的实例的子集的熵。

自定义信息增益函数IG()，计算结果中food\_cat\_food特征列的值0.463587为所有特征列里最小信息熵，因此用该特征列检测。

* 基尼不纯度（Gini impurity）

除了通过创建能产生最大信息增益的节点来创建一个决策树，还可以用启发性算法基尼不纯度（Gini impurity）衡量一个集合中类的比例，其公式为： *jt P(i | t)i*。如果分类问题包括两个类，Gini的最大值等于1/2。

sklearn库中DecisionTreeClassifier算法在参数criterion中，给出了上述两种方法{“gini”, “entropy”}，可以自行配置。

def entropy\_compomnent(numerator,denominator):   
 '''  
 function - 计算信息熵分量  
   
 Params:  
 numerator - 分子；  
 denominator - 分母；  
   
 Returns:  
 信息熵分量；float  
 '''  
 import math  
   
 if numerator!=0:  
 return -numerator/denominator\*math.log2(numerator/denominator)   
 elif numerator==0:  
 return 0  
  
print('层-A 根节点- 类标species(8,6):entropy={entropy}'.format(entropy=(entropy\_compomnent(6,14)+entropy\_compomnent(8,14))))  
print('层-B - food\_cat food(8,6):左子节点8(2,6)-Left\_entropy={L\_entropy};右子节点6(6,0)-Right\_entropy={R\_entropy};'.format(L\_entropy=(entropy\_compomnent(2,8)+entropy\_compomnent(6,8)),R\_entropy=(entropy\_compomnent(0,6)+entropy\_compomnent(6,6))))  
print('层-C - grumpy\_Yes-8(4,4):左子节点4(0,4)-Left\_entropy={L\_entropy};右子节点4(2,2)-Right\_entropy={R\_entropy};'.format(L\_entropy=(entropy\_compomnent(0,4)+entropy\_compomnent(4,4)),R\_entropy=(entropy\_compomnent(2,4)+entropy\_compomnent(2,4))))

层-A 根节点- 类标species(8,6):entropy=0.9852281360342516  
层-B - food\_cat food(8,6):左子节点8(2,6)-Left\_entropy=0.8112781244591328;右子节点6(6,0)-Right\_entropy=0.0;  
层-C - grumpy\_Yes-8(4,4):左子节点4(0,4)-Left\_entropy=0.0;右子节点4(2,2)-Right\_entropy=1.0;

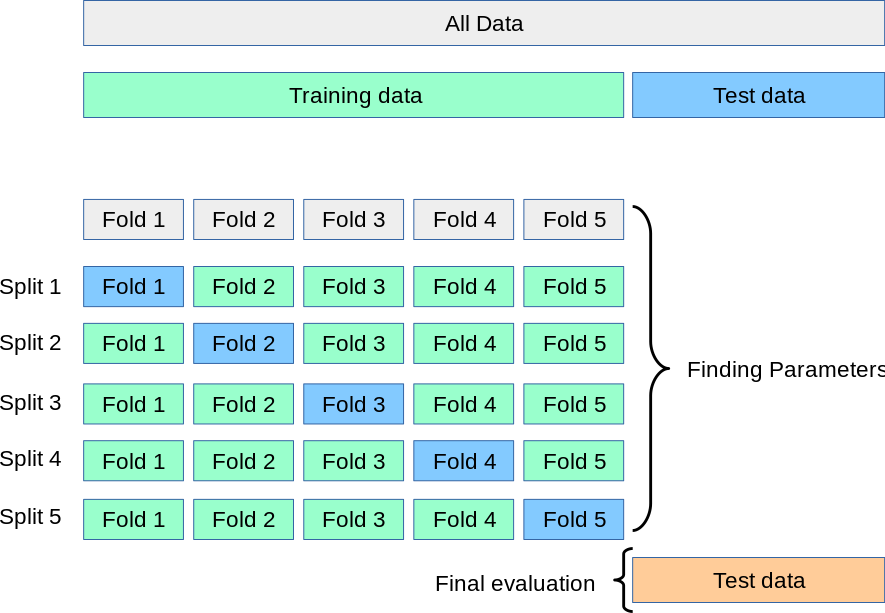
def IG(df\_dummies):   
 '''  
 function - 计算信息增量（IG）  
   
 Params:  
 df\_dummies - DataFrame格式，独热编码的特征值；DataFrame  
   
 Returns:  
 cal\_info\_df - 信息增益（Information gain）；DataFrame  
 '''  
 import pandas as pd   
   
 weighted\_frequency=df\_dummies.apply(pd.Series.value\_counts)  
 weighted\_sum=weighted\_frequency.sum(axis=0)   
 feature\_columns=weighted\_frequency.columns.tolist()  
 Parent\_entropy=entropy\_compomnent(weighted\_frequency[feature\_columns[-1]][0],14)+entropy\_compomnent(weighted\_frequency[feature\_columns[-1]][1],14)  
   
 cal\_info=[]  
 for feature in feature\_columns[:-2]:   
 v\_0\_frequency=df\_dummies.query('%s==0'%feature).iloc[:,-1].value\_counts().reindex(df\_dummies[feature].unique(),fill\_value=0) # 频数可能为0，如果为0则会被舍弃（value\_counts），因此需要补回（.reindex）  
 v\_1\_frequency=df\_dummies.query('%s==1'%feature).iloc[:,-1].value\_counts().reindex(df\_dummies[feature].unique(),fill\_value=0)  
 first\_child\_entropy=entropy\_compomnent(v\_0\_frequency[0], v\_0\_frequency.sum(axis=0))+entropy\_compomnent(v\_0\_frequency[1], v\_0\_frequency.sum(axis=0))   
 second\_child\_entropy=entropy\_compomnent(v\_1\_frequency[0], v\_1\_frequency.sum(axis=0))+entropy\_compomnent(v\_1\_frequency[1], v\_1\_frequency.sum(axis=0))  
  
 cal\_dic={'test':feature,  
 'Parent\_entropu':Parent\_entropy,  
 'first\_child\_entropy':first\_child\_entropy,  
 'second\_child\_entropy':second\_child\_entropy,  
 'Weighted\_average\_expression':'%f\*%d/%d+%f\*%d/%d'%(first\_child\_entropy,weighted\_frequency[feature][0],weighted\_sum.loc[feature],second\_child\_entropy,weighted\_frequency[feature][1],weighted\_sum.loc[feature]),  
 'IG':first\_child\_entropy\*(weighted\_frequency[feature][0]/weighted\_sum.loc[feature])+second\_child\_entropy\*(weighted\_frequency[feature][1]/weighted\_sum.loc[feature])  
   
 }  
 cal\_info.append(cal\_dic)  
 cal\_info\_df=pd.DataFrame.from\_dict(cal\_info)   
 return cal\_info\_df  
  
cal\_info\_df=IG(df\_dummies=catDog\_trainingData\_dummies,)  
util\_misc.print\_html(cal\_info\_df,row\_numbers=7)

|  | **test** | **Parent\_entropu** | **first\_child\_entropy** | **second\_child\_entropy** | **Weighted\_average\_expression** | **IG** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | play\_No | 0.985228 | 0.721928 | 0.764205 | 0.721928\*5/14+0.764205\*9/14 | 0.749106 |
| **1** | play\_Yes | 0.985228 | 0.764205 | 0.721928 | 0.764205\*9/14+0.721928\*5/14 | 0.749106 |
| **2** | grumpy\_No | 0.985228 | 0.811278 | 0.918296 | 0.811278\*8/14+0.918296\*6/14 | 0.857143 |
| **3** | grumpy\_Yes | 0.985228 | 0.918296 | 0.811278 | 0.918296\*6/14+0.811278\*8/14 | 0.857143 |
| **4** | food\_bacon | 0.985228 | 0.918296 | 0.970951 | 0.918296\*9/14+0.970951\*5/14 | 0.937101 |
| **5** | food\_cat\_food | 0.985228 | 0.811278 | 0.000000 | 0.811278\*8/14+0.000000\*6/14 | 0.463587 |
| **6** | food\_dog\_food | 0.985228 | 0.845351 | 0.000000 | 0.845351\*11/14+0.000000\*3/14 | 0.664204 |

* 使用sklearn库的DecisionTreeClassifier实现决策树及打印决策树流程图表。
* [交叉验证 （cross\_val\_score）](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)⑩

训练和测试数据集如果相同，即在相同的数据上训练和测试，模型（估计器，estimator）只会重复它刚刚看到的样本标签，可能会获得完美的分数，但无法正确预测其它的数据，这种情况称之为过拟合。因此在训练机器学习模型时，通常的做法是将数据集分为训练和测试数据集，但是手动配置超参（数）时，因为参数可以调整以使估计器达到最优，使得存在测试集过拟合的风险。这样，关于测试集的“知识”学习就会“泄露”到模型中，评估指标就不再报告泛化性能。为了解决这个问题，数据集被切分为训练数据集、验证数据集和测试数据集。然而，将数据集划分为三个集合，大大减少了学习模型的样本数量，及训练和验证数据集组合随机选择的机会。解决这一问题的方案可以应用交叉验证（cross val score，CV）的方法。

在CV方法中，测试数据集仍然用于最终的模型评估，但是不再需要验证数据集。被称为k-fold（k-倍/重/折）的CV中，数据集被分成k个更小的集合，对于每k个folds，k-1个folds用于训练集，1个fold用于测试集。得到的模型在剩余的数据上进行验证（也就是说，它被用作一个测试集来计算性能度量，比如精度）。如下图所示（颜色表示数据集类型）。

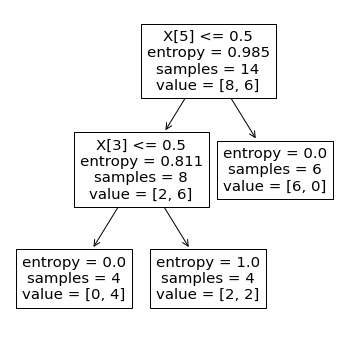


图片引自：[sklearn官网——Cross-validation: evaluating estimator performance](https://scikit-learn.org/stable/modules/cross_validation.html)⑪

X=catDog\_trainingData\_dummies[catDog\_trainingData\_dummies.columns[:-2]].to\_numpy()  
y=catDog\_trainingData\_df\_encoder[catDog\_trainingData\_df\_encoder.columns[-1]].to\_numpy()

def decisionTree\_structure(X,y,criterion='entropy',cv=None,figsize=(6, 6)):  
 '''  
 function - 使用决策树分类，并打印决策树流程图表。迁移于Sklearn的'Understanding the decision tree structure', https://scikit-learn.org/stable/auto\_examples/tree/plot\_unveil\_tree\_structure.html#sphx-glr-auto-examples-tree-plot-unveil-tree-structure-py  
   
 Params:  
 X - 数据集-特征值（解释变量）；ndarray  
 y- 数据集-类标/标签(响应变量)；ndarray  
 criterion - DecisionTreeClassifier 参数，衡量拆分的质量，即衡量哪一项检测最能减少分类的不确定性；string  
 cv - cross\_val\_score参数，确定交叉验证分割策略，默认值为None，即5-fole(折)的交叉验证；int  
   
 Returns:  
 clf - 返回决策树模型  
 '''  
 import numpy as np  
 from matplotlib import pyplot as plt  
  
 from sklearn.model\_selection import train\_test\_split  
 from sklearn.datasets import load\_iris  
 from sklearn.tree import DecisionTreeClassifier  
 from sklearn import tree  
 from sklearn.model\_selection import cross\_val\_score   
   
 #X\_train, X\_test, y\_train, y\_test=train\_test\_split(X, y, random\_state=0)  
 X\_train,y\_train=X,y  
 clf=DecisionTreeClassifier(criterion=criterion,max\_leaf\_nodes=3, random\_state=0)  
 clf.fit(X\_train, y\_train)   
   
 n\_nodes=clf.tree\_.node\_count  
 children\_left=clf.tree\_.children\_left  
 children\_right=clf.tree\_.children\_right  
 feature=clf.tree\_.feature  
 threshold=clf.tree\_.threshold   
 print("n\_nodes:{n\_nodes},\nchildren\_left:{children\_left},\nchildren\_right={children\_right},\nthreshold={threshold}".format(n\_nodes=n\_nodes,children\_left=children\_left,children\_right=children\_right,threshold=threshold))  
 print("\_"\*50)  
   
 node\_depth=np.zeros(shape=n\_nodes, dtype=np.int64)  
 is\_leaves=np.zeros(shape=n\_nodes, dtype=bool)  
 stack=[(0, 0)] # start with the root node id (0) and its depth (0)   
  
 while len(stack) > 0:  
 # `pop` ensures each node is only visited once  
 node\_id, depth = stack.pop()  
 node\_depth[node\_id] = depth  
  
 # If the left and right child of a node is not the same we have a split  
 # node  
 is\_split\_node = children\_left[node\_id] != children\_right[node\_id]  
 # If a split node, append left and right children and depth to `stack`  
 # so we can loop through them  
 if is\_split\_node:  
 stack.append((children\_left[node\_id], depth + 1))  
 stack.append((children\_right[node\_id], depth + 1))  
 else:  
 is\_leaves[node\_id] = True  
  
 print("The binary tree structure has {n} nodes and has "  
 "the following tree structure:\n".format(n=n\_nodes))  
 for i in range(n\_nodes):  
 if is\_leaves[i]:  
 print("{space}node={node} is a leaf node.".format(  
 space=node\_depth[i] \* "\t", node=i))  
 else:  
 print("{space}node={node} is a split node: "  
 "go to node {left} if X[:, {feature}] <= {threshold} "  
 "else to node {right}.".format(  
 space=node\_depth[i] \* "\t",  
 node=i,  
 left=children\_left[i],  
 feature=feature[i],  
 threshold=threshold[i],  
 right=children\_right[i]))   
   
 plt.figure(figsize=figsize)   
 tree.plot\_tree(clf)  
 plt.show()  
   
 CV\_scores=cross\_val\_score(clf,X,y, cv=cv)  
 print('cross\_val\_score:\n',CV\_scores) # 交叉验证每次运行的估计器得分数组   
 print("%0.2f accuracy with a standard deviation of %0.2f" % (CV\_scores.mean(), CV\_scores.std())) # 同时给出了平均得分，和标准差  
 return clf  
   
clf=decisionTree\_structure(X,y)

n\_nodes:5,  
children\_left:[ 1 3 -1 -1 -1],  
children\_right=[ 2 4 -1 -1 -1],  
threshold=[ 0.5 0.5 -2. -2. -2. ]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The binary tree structure has 5 nodes and has the following tree structure:  
  
node=0 is a split node: go to node 1 if X[:, 5] <= 0.5 else to node 2.  
 node=1 is a split node: go to node 3 if X[:, 3] <= 0.5 else to node 4.  
 node=2 is a leaf node.  
 node=3 is a leaf node.  
 node=4 is a leaf node.



cross\_val\_score:  
 [0.66666667 0.66666667 0.66666667 1. 0.5 ]  
0.70 accuracy with a standard deviation of 0.16

#### 2）随机森林（Random forests）

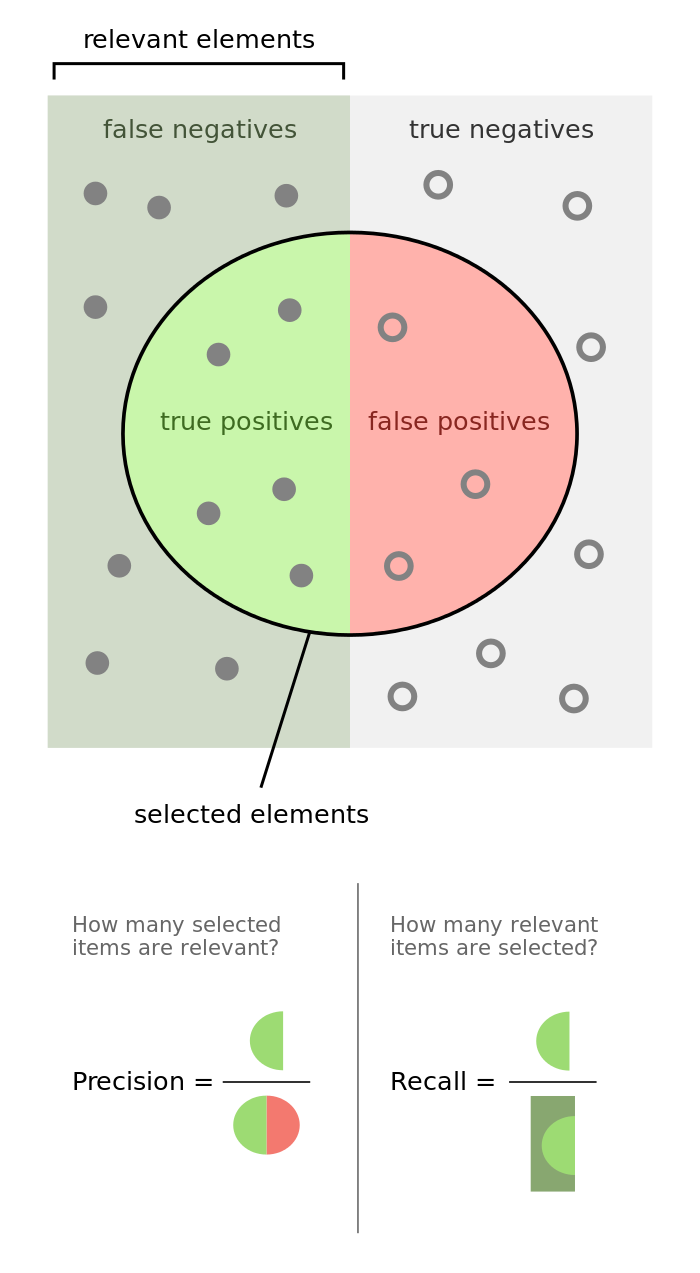
决策树属于勤奋学习模型（eager learners），与之相反的是KNN算法这样的惰性学习模型（lazy learners）。决策树学习算法会产生出完美拟合每一个训练实例的巨型复杂的决策树模型，而无法对真实的关系进行泛化，即容易过拟合。解决决策树过拟合的方法可以通过剪枝的方法移除决策树中过深的节点和叶子，或者从训练数据和特征的子集中创建多棵决策树构成多个模型的集成（集成是指多个估计器的组合），被称为随机森林的决策树集合。创建集成的方法有套袋法（bagging），推进法（boosting）和堆叠法（stacking）。

* 分类报告（classification\_report）

sklearn.metrics.classification\_report用于显示分类指标的文本报告，在报告中显示有每个类的精确度（precision），召回率（recall），F1分数（f1-score）等信息。精确度和召回率可以从Wikipedia的[Precision and recall](https://en.wikipedia.org/wiki/Precision_and_recall)⑫中获取详细的解释，给出的阐释图表可以一目了然的理解计算方法。实心小圆代表类标狗总计12只，空心小圆代表类标猫总计10只。通过假设的模型预测，得到正确预测为狗的有5只，正确预测为猫的为3只，其它的均为错误预测。因此对于分类狗而言， precision=5/8 recall=5/12 。

support字段为每个标签出现的次数。avg行为均值和加权均值（参数sample\_weight 配置权重值，默认为None），avg行对应的support为标签总和。

从计算结果可知，使用随机森林算法的各项指标均高于决策树算法的结果。



图片引自：[Wikipedia——Precision and recall](https://en.wikipedia.org/wiki/Precision_and_recall)

from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report  
  
X,y=make\_classification(n\_samples=1000,n\_features=100,n\_informative=20,n\_clusters\_per\_class=2,random\_state=11)  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,random\_state=11)  
clf\_DTC=DecisionTreeClassifier(random\_state=11)  
clf\_DTC.fit(X\_train,y\_train)  
predictions\_DTC=clf\_DTC.predict(X\_test)  
print(classification\_report(y\_test,predictions\_DTC))

precision recall f1-score support  
  
 0 0.73 0.66 0.69 127  
 1 0.68 0.75 0.71 123  
  
 accuracy 0.70 250  
 macro avg 0.71 0.70 0.70 250  
weighted avg 0.71 0.70 0.70 250

clf\_RFC=RandomForestClassifier(n\_estimators=10,random\_state=11)  
clf\_RFC.fit(X\_train,y\_train)  
predictions\_RFC=clf\_RFC.predict(X\_test)  
print(classification\_report(y\_test,predictions\_RFC))

precision recall f1-score support  
  
 0 0.74 0.83 0.79 127  
 1 0.80 0.70 0.75 123  
  
 accuracy 0.77 250  
 macro avg 0.77 0.77 0.77 250  
weighted avg 0.77 0.77 0.77 250

### 2.5.2.5 图像分类\_识别器

#### 1）图像分类器

应用极端随机森林（extremely randomized trees, Extra-Trees）训练分类模型，注意应用了preprocessing.LabelEncoder()方法编码，如果要映射回原来的类标，可以执行.inverse\_transform实现。从对估计器的评测结果来看，f1-score的平均得分为0.62，分项得分中“开阔”、“窄建”和“阔建”分类的预测得分大于0.7，相对较好；而“林荫”、“窄木”、“宽建”和“宽木”都小于0.5，预测精度并不理想。这里的一个主要原因是图像分类的确定，这几个分类中有很多图像并不容易区分之间的差异，也就导致了对图像的分类选择并不专业，最终致使建立的预测模型的预测精度并不是很好，可以尝试从新建立分类标准，提供数据集的类标精度。

class ERF\_trainer:  
 '''  
 class - 用极端随机森林训练图像分类器  
 '''  
 def \_\_init\_\_(self,X,label\_words,save\_path):  
 from sklearn import preprocessing  
 from sklearn.ensemble import ExtraTreesClassifier  
 import os,pickle  
   
 print('Start training...')  
 self.le=preprocessing.LabelEncoder()  
 self.clf=ExtraTreesClassifier(n\_estimators=100,max\_depth=16,random\_state=0) # http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html  
 y=self.encode\_labels(label\_words)  
 self.clf.fit(np.asarray(X),y)  
 with open(os.path.join(save\_path,'ERF\_clf.pkl'), 'wb') as f: # 存储训练好的图像分类器模型  
 pickle.dump(self.clf, f)   
 print("end training and saved estimator.")  
   
 def encode\_labels(self,label\_words):  
 '''  
 function - 对标签编码，及训练分类器  
 '''  
 self.le.fit(label\_words)  
 return np.array(self.le.transform(label\_words),dtype=np.float64)  
   
 def classify(self,X):  
 '''  
 function - 对未知数据的预测分类  
 '''  
 label\_nums=self.clf.predict(np.asarray(X))  
 label\_words=self.le.inverse\_transform([int(x) for x in label\_nums])  
 return label\_words  
   
import pickle  
import numpy as np   
  
feature\_map\_fp='./model/visual\_feature.pkl'   
with open(feature\_map\_fp,'rb') as f:  
 feature\_map=pickle.load(f) # 读取存储的图像特征  
label\_words=[x['object\_class'] for x in feature\_map]   
dim\_size=feature\_map[0]['feature\_vector'].shape[1]   
X=[np.reshape(x['feature\_vector'],(dim\_size,)) for x in feature\_map]  
save\_path='./model'  
  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test=train\_test\_split(X, label\_words,test\_size=0.3, random\_state=42)  
erf=ERF\_trainer(X\_train,y\_train,save\_path)

Start training...  
end training and saved estimator.

from sklearn.metrics import classification\_report  
print(classification\_report(y\_test,erf.classify(X\_test)))

precision recall f1-score support  
  
 宽建 0.44 0.21 0.29 19  
 宽木 0.55 0.39 0.46 28  
 开阔 1.00 0.68 0.81 28  
 林荫 0.20 0.17 0.18 12  
 窄建 0.74 0.93 0.83 61  
 窄木 0.25 0.09 0.13 22  
 阔建 0.88 0.64 0.74 11  
 阔木 0.54 0.81 0.65 59  
  
 accuracy 0.62 240  
 macro avg 0.57 0.49 0.51 240  
weighted avg 0.61 0.62 0.59 240

#### 2）图像识别器

图像识别器需要调用3个已经保存的文件，第一个是“ERF\_clf.pkl”，保存有应用极端随机森林算法训练的图像分类器模型，用于类别预测；第二个是“visual\_BOW.pkl”，保存有视觉词袋KMeans聚类模型，用于构建图像特征（位于前述的feature\_builder\_BOW类中）的参数输入；第三个是“visual\_feature.pkl”，存储的是图像特征，主要是读取类标转换为整型编码，然后应用inverse\_transform方法将预测的整型编码转换为原始的类标。

class ImageTag\_extractor:  
 '''  
 class - 图像识别器，基于图像分类模型，视觉词袋以及图像特征  
 '''  
 def \_\_init\_\_(self, ERF\_clf\_fp, visual\_BOW\_fp,visual\_feature\_fp):  
 from sklearn import preprocessing  
 import pickle   
 with open(ERF\_clf\_fp,'rb') as f: # 读取存储的图像分类器模型  
 self.clf=pickle.load(f)  
  
 with open(visual\_BOW\_fp,'rb') as f: # 读取存储的聚类模型和聚类中心点  
 self.kmeans=pickle.load(f)  
  
 '''对标签编码'''  
 with open(visual\_feature\_fp, 'rb') as f:  
 self.feature\_map=pickle.load(f)  
 self.label\_words=[x['object\_class'] for x in self.feature\_map]  
 self.le=preprocessing.LabelEncoder()  
 self.le.fit(self.label\_words)   
   
 def predict(self,img):  
 import util\_A  
 import numpy as np  
 feature\_vector=util\_A.feature\_builder\_BOW().construct\_feature(img,self.kmeans) # 提取图像特征，之前定义的feature\_builder\_BOW()类，可放置于util.py文件中，方便调用  
 label\_nums=self.clf.predict(np.asarray(feature\_vector)) # 进行图像识别/分类  
 image\_tag=self.le.inverse\_transform([int(x) for x in label\_nums])[0] # 获取图像分类标签  
 return image\_tag  
   
ERF\_clf\_fp=r'./model/ERF\_clf.pkl'  
visual\_BOW\_fp='./model/visual\_BOW.pkl'  
visual\_feature\_fp='./model/visual\_feature.pkl'  
  
import cv2 as cv  
import os  
imgs\_fp=r'C:\Users\richi\omen\_richiebao\omen\_github\USDA\_CH\_final\USDA\notebook\data\kitti'  
imgs\_=[os.path.join(imgs\_fp,f) for f in os.listdir(imgs\_fp)]  
imgs\_pred\_tag={fn:ImageTag\_extractor(ERF\_clf\_fp,visual\_BOW\_fp,visual\_feature\_fp).predict(cv.imread(fn)) for fn in imgs\_}

在GoogleEarth的街景（Street view）德国随机城市下，随机的截取了6张尺寸大小不一的图像，应用图像识别器预测分类，其结果如下。

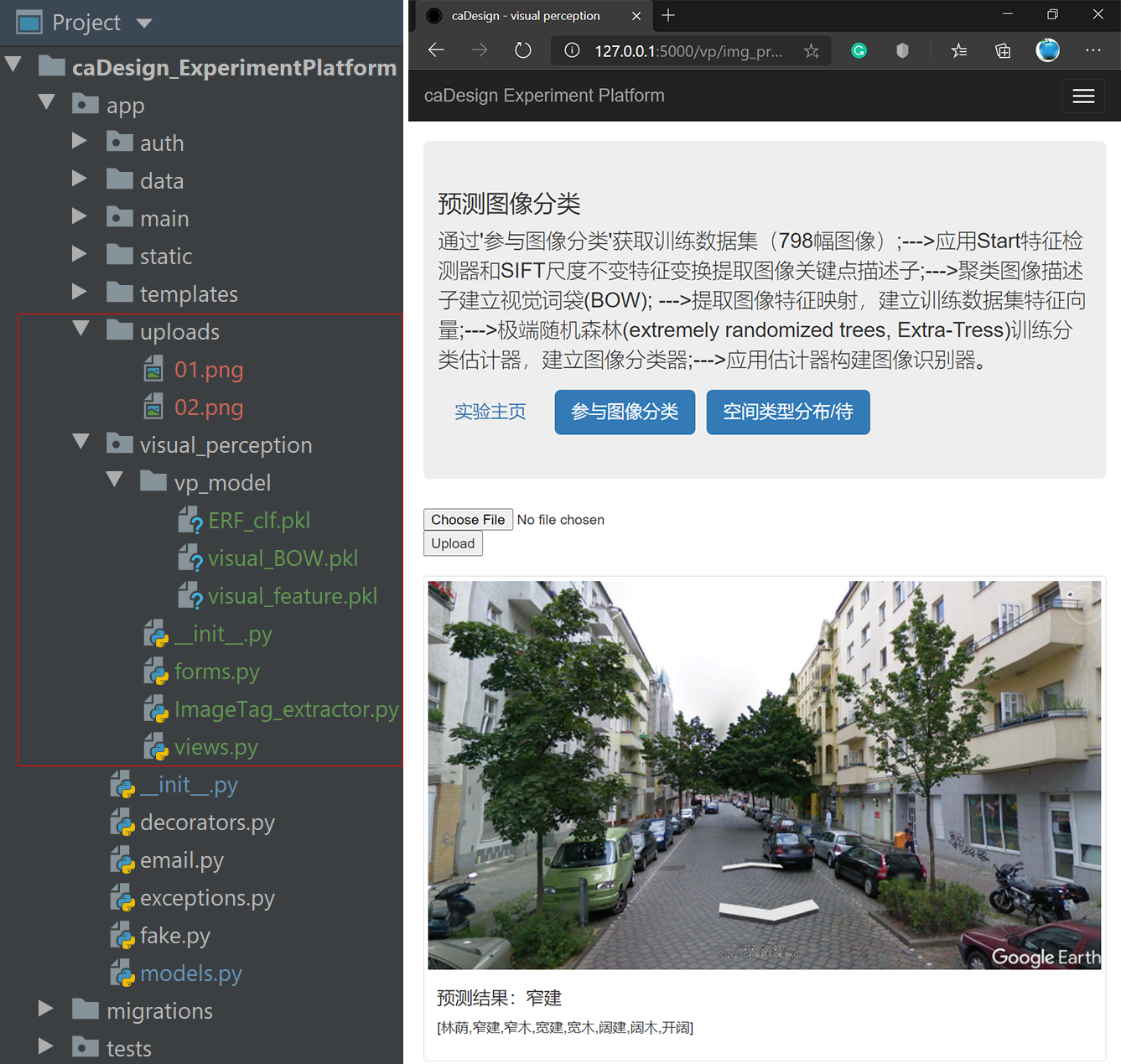
import matplotlib.pyplot as plt  
from PIL import Image  
import matplotlib  
matplotlib.rcParams['font.family'] = ['SimHei']  
  
fig,axes=plt.subplots(2,3,sharex=True,sharey=True,figsize=(25,10)) # 布局多个子图，每个子图显示一幅图像  
ax=axes.flatten() # 降至1维，便于循环操作子图  
i=0  
for f,tag in imgs\_pred\_tag.items():  
 img\_array=Image.open(f)  
 ax[i].imshow(img\_array) # 显示图像  
 ax[i].set\_title("pred:{tag}".format(tag=tag))  
 i+=1  
fig.tight\_layout() # 自动调整子图参数，使之填充整个图像区域  
fig.suptitle("images show",fontsize=14,fontweight='bold',y=1.02)  
plt.show()



#### 3）嵌入图像识别器到网络实验平台

将估计器部署到网络实验平台，需要将图像识别器及其相关的代码和文件（估计器、视觉词袋和图像特征）整合起来。下述将ImageTag\_extractor和feature\_builder\_BOW类置于同一个文件中(app/visual\_perception/ImageTag\_extractor.py)，同时将feature\_builder\_BOW类中不需要的功能移除，保持代码简洁，避免干扰，并增加了ImageTag\_extractor\_execution类，方便外部调用该类，直接执行预测。而“ERF\_clf.pkl”，“visual\_BOW.pkl”和“visual\_feature.pkl”三个必需文件置于文件夹visual\_perception/vp\_model中。

预测图像分类的功能是可以在页面端上传一幅图像后，用图像识别器（已经训练好的图像分类器）预测分类，因此增加了一个新的文件夹uploads用于保存上传的图像文件。图像上传的配置主要使用flask\_uploads库实现，需要在app/\_\_init\_\_.py文件中增加相应的配置。



嵌入到网络实验平台的图像识别器

# app/visual\_perception/ImageTag\_extractor.py  
class feature\_builder\_BOW:  
 '''  
 class - (仅保留construct\_feature及关联部分函数)根据所有图像关键点描述子聚类建立图像视觉词袋，获取每一图像的特征（码本）映射的频数统计  
 '''  
  
 def \_\_init\_\_(self, num\_cluster=32):  
 self.num\_clusters = num\_cluster  
  
 def extract\_features(self, img):   
 '''  
 function - 提取图像特征  
  
 Paras:  
 img - 读取的图像  
 '''  
 import cv2 as cv  
   
 star\_detector = cv.xfeatures2d.StarDetector\_create()  
 key\_points = star\_detector.detect(img)  
 img\_gray = cv.cvtColor(img, cv.COLOR\_BGR2GRAY)  
 kp, des = cv.xfeatures2d.SIFT\_create().compute(img\_gray, key\_points) # SIFT特征提取器提取特征  
 return des  
  
 def normalize(self, input\_data):   
 '''  
 fuction - 归一化数据  
  
 input\_data - 待归一化的数组  
 '''  
 import numpy as np  
   
 sum\_input = np.sum(input\_data)  
 if sum\_input > 0:  
 return input\_data / sum\_input # 单一数值/总体数值之和，最终数值范围[0,1]  
 else:  
 return input\_data  
  
 def construct\_feature(self, img, kmeans):   
 '''  
 function - 使用聚类的视觉词袋构建图像特征（构造码本）  
  
 Paras:  
 img - 读取的单张图像  
 kmeans - 已训练的聚类模型  
 '''  
 import numpy as np  
   
 des = self.extract\_features(img)  
 labels = kmeans.predict(des.astype(np.float)) # 对特征执行聚类预测类标  
 feature\_vector = np.zeros(self.num\_clusters)  
 for i, item in enumerate(feature\_vector): # 计算特征聚类出现的频数/直方图  
 feature\_vector[labels[i]] += 1  
 feature\_vector\_ = np.reshape(feature\_vector, ((1, feature\_vector.shape[0])))  
 return self.normalize(feature\_vector\_)  
  
class ImageTag\_extractor:  
 '''  
 class - 图像识别器，基于图像分类模型，视觉词袋以及图像特征  
 '''  
  
 def \_\_init\_\_(self, ERF\_clf\_fp, visual\_BOW\_fp, visual\_feature\_fp):  
 from sklearn import preprocessing  
 import pickle  
 with open(ERF\_clf\_fp, 'rb') as f: # 读取存储的图像分类器模型  
 self.clf = pickle.load(f)  
  
 with open(visual\_BOW\_fp, 'rb') as f: # 读取存储的聚类模型和聚类中心点  
 self.kmeans = pickle.load(f)  
  
 '''对标签编码'''  
 with open(visual\_feature\_fp, 'rb') as f:  
 self.feature\_map = pickle.load(f)  
 self.label\_words = [x['object\_class'] for x in self.feature\_map]  
 self.le = preprocessing.LabelEncoder()  
 self.le.fit(self.label\_words)  
  
 def predict(self, img):  
 import numpy as np  
 feature\_vector=feature\_builder\_BOW().construct\_feature(img, self.kmeans) # 提取图像特征，之前定义的feature\_builder\_BOW()类，可放置于util.py文件中，方便调用  
 label\_nums = self.clf.predict(np.asarray(feature\_vector)) # 进行图像识别/分类  
 image\_tag = self.le.inverse\_transform([int(x) for x in label\_nums])[0] # 获取图像分类标签  
 return image\_tag  
  
  
class ImageTag\_extractor\_execution:  
 def \_\_init\_\_(self,img\_url):  
 self.img\_url=img\_url  
 self.ERF\_clf\_fp='app/visual\_perception/vp\_model/ERF\_clf.pkl'  
 self.visual\_BOW\_fp = 'app/visual\_perception/vp\_model/visual\_BOW.pkl'  
 self.visual\_feature\_fp = 'app/visual\_perception/vp\_model/visual\_feature.pkl'  
  
 def execution(self):  
 import cv2 as cv  
 print("\*"\*50)  
 print(self.img\_url)  
 imgs\_pred\_tag=ImageTag\_extractor(self.ERF\_clf\_fp, self.visual\_BOW\_fp, self.visual\_feature\_fp).predict(cv.imread(self.img\_url))  
 return imgs\_pred\_tag

配置flask\_uploads。

# app/\_\_init\_\_.py  
#...  
from flask\_uploads import UploadSet, configure\_uploads, IMAGES, patch\_request\_class  
import os  
#...  
photos=UploadSet('photos', IMAGES)  
  
def create\_app(config\_name):  
 #...  
 basedir=os.path.abspath(os.path.dirname(\_\_file\_\_))  
 app.config['UPLOADED\_PHOTOS\_DEST']=os.path.join(basedir,'uploads') # you'll need to create a folder named uploads  
 configure\_uploads(app, photos)  
 patch\_request\_class(app)  
 #...

配置图像文件上传Web表达。

#app/visual\_perception/forms.py  
#...  
from .. import photos  
#...  
class upload\_img(FlaskForm):  
 photo=FileField(validators=[FileAllowed(photos, 'Image only!'), FileRequired('File was empty!')])  
 submit=SubmitField('Upload')

配置路由和视图函数，在视图函数中调用图像识别器。

#app/visual\_perception/views.py  
#...  
from .. import photos  
from .ImageTag\_extractor import ImageTag\_extractor\_execution  
#...  
@visual\_perception.route("/img\_prediction",methods=['GET','POST'])  
def img\_prediction():  
 import os  
 form=upload\_img()  
 if form.validate\_on\_submit():  
 img\_name=photos.save(form.photo.data)  
 img\_url=photos.url(img\_name)  
 img\_fp=os.path.join(current\_app.config['UPLOADED\_PHOTOS\_DEST'],img\_name)  
 imgs\_pred\_tag=ImageTag\_extractor\_execution(img\_fp).execution()  
 return render\_template('vp/img\_prediction.html', form=form, img\_url=img\_url, imgs\_pred\_tag=imgs\_pred\_tag)  
 else:  
 img\_url=None  
 return render\_template('vp/img\_prediction.html', form=form, img\_url=img\_url)

配置“预测图像分类”的模板页面

<!--templates/vp/img\_prediction.html -->   
{% extends "base.html" %}  
{% import "bootstrap/wtf.html" as wtf %}  
{% import "\_macros.html" as macros %}  
  
{% block title %}caDesign - visual perception{% endblock %}  
  
{% block page\_content %}  
 <div class="jumbotron">  
 <h3>预测图像分类</h3>  
  
 <p>  
 通过'参与图像分类'获取训练数据集（798幅图像）;--->应用Start特征检测器和SIFT尺度不变特征变换提取图像关键点描述子;--->聚类图像描述子建立视觉词袋(BOW);  
 --->提取图像特征映射，建立训练数据集特征向量;--->极端随机森林(extremely randomized trees, Extra-Tress)训练分类估计器，建立图像分类器;--->应用估计器构建图像识别器。  
 </p>  
  
 <p>  
 <a class="btn btn-secondary btn-lg" href="{{ url\_for('visual\_perception.vp') }}" role="button">实验主页</a>  
 &nbsp<a class="btn btn-primary btn-lg" href="{{ url\_for('visual\_perception.imgs\_classification') }}" role="button">参与图像分类</a>  
 &nbsp<a class="btn btn-primary btn-lg" href="{{ url\_for('visual\_perception.vp') }}" role="button">空间类型分布/待</a>  
 </p>  
 </div>  
  
 <form method="POST" enctype="multipart/form-data">  
 {{ form.hidden\_tag() }}  
 {{ form.photo }}  
 {% for error in form.photo.errors %}  
 <span style="color: red;">{{ error }}</span>  
 {% endfor %}  
 {{ form.submit }}  
 </form>  
  
 {% if img\_url %}  
 <br>  
 <div class="thumbnail">  
 <img src="{{ img\_url }}" >  
 <div class="caption">  
 <h4>预测结果：{{ imgs\_pred\_tag }}</h4>  
 <p>[林荫,窄建,窄木,宽建,宽木,阔建,阔木,开阔] </p>  
 </div>  
 </div>  
 {% endif %}  
  
{% endblock %}

#### 4）部署

在服务器（Linux系统）下部署基于Flask构建的网络实验平台，其基本流程是：

1. 建立Python虚拟环境，并根据生成的“requirements.txt”文件安装库配置环境；
2. 安装nginx，uwsgi，flask；
3. 建立一个简单的App应用，测试程序；
4. 本地项目上传至服务器虚拟环境目录下；
5. 建立uWSGI入口点（entry points）；
6. 配置uWSGI；
7. 建立Upstart Script；
8. 配置Nginx；8.supervisor进程守护程序。

虽然网络上可以搜索到大量在Linux系统服务器部署Flask的教程，但是因为系统可能会存在不同，说明文字上的不清晰，Flask文件结构的差异，Python环境的变化，尤其库版本的变化，通常在部署时很难一帆风顺。即使给出的配置步骤没有问题，部署的过程也会容易出现不经意的错误，因此需要耐心的跟随步骤操作，同时最好用markdown工具将步骤，所用到的命令记录下来，方便查看。具体可以尝试参考[How To Serve Flask Applications with uWSGI and Nginx on Ubuntu 14.04](https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-uwsgi-and-nginx-on-ubuntu-14-04)⑬或者自行搜索。

注释（Notes）：

① WIX，创建、设计、管理自定义网站开发平台（<https://www.wix.com/>）。

② Flask，用Python编写的微型网络框架，为微框架（microframework），因为它没有数据库抽象层、表单验证或其他任何预先存在的第三方库提供通用功能的组件。然而，Flask支持扩展，可以增加应用功能，例如对象关系映射器、表单验证、上传处理、各种开放的认证技术和一些常见的框架相关工具都有扩展[（<https://pypi.org/project/Flask/>）。

③ Flask推荐阅读官方文档，（<https://flask.palletsprojects.com/en/2.2.x/>）。

④ caDesign\_ExperimentPlatform，项目实验GitHub仓库地址（<https://github.com/richieBao/caDesign_ExperimentPlatform>）。

⑤ PyCharm，用于计算机编程的集成开发环境(IDE)，主要用于Python语言开发，提供代码分析、图形化调试器，集成测试器、集成版本控制系统，并支持使用Django进行网页开发（<https://www.jetbrains.com/pycharm/>）。

⑥ Visual Paradigm（VP），一套设计、分析和管理工具，推动IT项目开发和数字化转型，包括代码逆向工程（<https://www.visual-paradigm.com/>）。

⑦ Pyreverse，一套用于逆向工程的Python代码的工具，使用一个Python项目的类层次结构表示，可以用来提取任何信息（如生成UML图或单元测试，如pyargo和py2tests）（<https://pypi.org/project/pyreverse/>）。

⑧ Jinjia，仿照Django模板，一种现代的、对设计师友好的Python模板语言（引擎），其使用范围广，而且有可选择的沙盒式模板（sandboxed template）执行环境，快速安全（<https://jinja.palletsprojects.com/en/2.11.x/>）。

⑨ Sourcetrail，是一个免费和开源的跨平台源码资源管理器，支持C、C++、Java和Python编程语言，可视化动态生成任何选定类型、函数、变量等的交互式代码映射，并显示与其余代码库的所有依存关系。（<https://github.com/CoatiSoftware/Sourcetrail>）。

⑩ 交叉验证 （cross\_val\_score），通过交叉验证来评估一个分数（<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html>）。

⑪ sklearn官网——Cross-validation: evaluating estimator performance，评估估计器的性能（<https://scikit-learn.org/stable/modules/cross_validation.html>）。

⑫ Precision and recall，在模式识别（pattern recognition）、信息检索（information retrieval）、对象检测（object detection）和分类（机器学习）中，精确度和召回率是适用于从集合、语料库或样本空间检索的数据的性能指标。（<https://en.wikipedia.org/wiki/Precision_and_recall>）。

⑬ How To Serve Flask Applications with uWSGI and Nginx on Ubuntu 14.04，（<https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-uwsgi-and-nginx-on-ubuntu-14-04>）。

参考文献（References）:

[1] Miguel Grinberg. Flask Web Development:Developing Web Applications with Python[M]. USA:O’Reilly Media, Inc.,2018. 2th Edition.

[2] Cavin Hackeling. Mastering Machine Learning with scikit-learn[M].Packt Publishing Ltd.July 2017.Second published.（Cavin Hackeling.张浩然译.scikit-learning 机器学习[M].人民邮电出版社.2019.2.）