《基于Python的Web服务框架搭建》说明文档

0. 运行环境及项目框架

本次作业代码在GNU/Linux下的Python 3.9.2中运行，需要以下软件包：

```
matplotlib=3.5.2
torch==1.12.0+cpu
torchvision==0.13.0+cpu
django==4.0.6
```

在src下主要有以下内容：

```
# src
> app
  > migrations
  __init__.py
  admin.py
  apps.py
  models.py   # SQL fields: model_name uploader created_at updated_at
  tests.py
  views.py
> src
  > data
    > MNIST
      > raw   # training data
  > models
    __init__.py
    lenet.py
  > utils
    pre_process.py
  __init__.py
  asgi.py
  launcher.py # launch the process of training and record time
  settings.py
  train.py
  urls.py     # URL mapping
  views.py    # functions: home train submit(for train.html) model records
  wsgi.py
> static
  > models
    > YYYYMMDDHHmmss
      > images
        accuracy.png
        loss.png
      YYDDMMHHmmss_lenet.pth
      output.log
> templates
  home.html
  model.html
  records.html
  train.html
db.sqlite3 # database
manage.py
```

除了基本的Django框架外，本次作业中还有与人工智能相关的代码及及训练、测试数据，即src中的train.py,models,utils和data文件夹；用于训练结果展示的数据文件在static/models中，以提交时间作为文件夹名，下面依次存放图片文件夹、训练好的模型文件以及训练日志文件；网页html模板在templates下，分别对应主页、训练页面、结果展示页面和任务详情页面；网站中用到的数据库由app文件夹中的应用文件实现，models.py中指定数据库字段为model_name,uploader,created_at,updated_at，而数据由db.sqlite3存储。

1. 主页的实现

主页所含的元素较少而结构简单，实现起来较为容易。脚注中的个人主页和邮箱链接可能是需要日后维护修改的，故这里实现为Django模板的形式。这里的mailto:是href中的一个特殊链接标记，表示用默认的邮箱客户端打开邮箱链接。

```
Home: <a href="{{ home }}">{{ home }}</a><br/>
Contact: <a href="mailto:{{ contact }}">{{ contact }}</a>
```

主页跳转到其他页面的功能使用button的onclick事件内嵌JavaScript代码来实现。这里的train页面逻辑较为简单，直接指定location.href=('train')即可，而records页面后续需要实现搜索和分类的功能，故此时需要在链接中指定正确的参数。records?model_name=&uploader=&page_num=1即表示将model_name和uploader这两个过滤条件置空，显示所有的记录，并定位到记录的第一页。

```
<button type="button" onclick="location.href=('train')">
  Start training your model
</button>
<button type="button"
        onclick="location.href=('records?model_name=&uploader=&page_num=1')">
  View previous results
</button>
```

在views.py中，主页的视图控制代码实现为home函数：

```
def home(request):
    context = {}
    context['home'] = "https://github.com/HongYurui"
    context['contact'] = "hongyr20@mails.tsinghua.edu.cn"
    return render(request, 'home.html', context)
```

这里，函数指定了主页和邮箱链接的具体内容，发送给home.html，实现页面的渲染和显示。

2. 训练页面的实现

在训练页面中，主要需要进行的是参数的检查和传递，以及训练子进程的开启，这意味着views.py中除了渲染初始页面的train函数外，需要实现一个较为复杂的submit函数来执行相应的检查和提交。

在输入的各个字段中，任务名称model_name和发起人uploader是数字、字母和下划线组成的字符串，是否添加一层卷积层、采用何种优化器在逻辑上是一个单选，学习率是一个0-1之间的整数，批次大小和训练总轮次为正整数。对于这些不同类型的输入，一方面，在html层面可以先行检查它们是否为空，即在输入框的标签种加入required属性，例如：

```
<input type="text" name="model_name" id="model_name"
      value="{{ model_name }}" required/>
```

这样就把输入框设置成了必填项，当用户试图在未填写完所有字段的情况下提交时，则会收到页面的提醒。

对于“连续变化”的模型参数学习率、批次大小和训练总轮次，我们希望设置一个默认值，给用户自己设定参数提供借鉴，这里就可以运用Django过滤器来设置默认值，例如：

```
<input type="text" name="learning_rate" id="learning_rate"
       value="{{ learning_rate|default:'0.01' }}" required>
```

这里，若提供的学习率值非空，则正常显示，否则默认为0.01，这样就实现了在用户尚未输入时指定字段的默认值，而没有把这一html层面的职能显式地加到views.py中。

不过，进一步判断输入合法性需要使用Python代码来实现，这里采用正则表达式来判断字符串的，用类型转换的异常处理来判断数值，而bool逻辑由于单选框的互斥且非空，已被实现，不用再考虑。

```
# regular expression for model name and uploader
for name in ['model_name', 'uploader']:
    if not re.match('^\\w+$', context[name]):
        context['error_' + name] =
            "Only letters, numbers and underscores are allowed"
        break
# exception handling for numbers, taking learning rate as example
try:
    learning_rate = float(learning_rate)
    assert learning_rate > 0 and learning_rate < 1
except (ValueError, AssertionError):
    context['error_learning_rate'] =
        "Learning rate must be a float between 0 and 1!"
```

对于不符合要求的输入，不仅不再触发正常的提交流程，而且在context中添加相应的错误提示，同时还要保证页面中已填写的其他信息在页面上仍然可见。因此，当按下提交按钮后，处理函数需要将页面提交的所有已输入信息和生成错误信息，放入context中，然后返回render函数，以便渲染页面。由于错误信息默认为空，因此，页面检查到错误，中止提交并返回错误信息，当且仅当至少一个错误信息不为空。

```
saved_data:dict = {}
for key in context.keys():
    if re.match("^error_\\w*", key):
        if context[key] != "":
            return render(request, 'train.html', context)
    else:
        saved_data[key] = context[key]
```

如果所有字段都合法，则在数据库中添加当前模型的记录，由任务名称、上传者，均设置为当前时间的发起时间和最后改动时间四个字段组成，并开启一个新进程，将所有的模型参数提供给它来进行训练。

```
# save to database
timestamp = datetime.datetime.now().replace(microsecond=0)
models.Record.objects.create(model_name=model_name, uploader=uploader,
```

```
    created_at=timestamp, updated_at=timestamp)
saved_data['time'] = timestamp

# begin training
os.popen("python3 src/launcher.py --args \""+str(saved_data)+"\"")

# redirect to records page
return redirect('/records?model_name=&uploader=&page_num=1')
```

这里的launcher.py除了调用train.py中的train函数进行训练外，还实现了一些必要的其他功能。在对输入参数进行解析后，launcher.py首先创建必要的文件路径及日志文件，然后开始训练并保存结果。训练结束后，launcher.py在数据库中更新该任务的修改时间，标记了任务的结束，以供前端刷新任务状态及完成时间。

```
# parse parameters
arg = argparse.ArgumentParser()
arg.add_argument('--args', type=str, default=None)
args_dict = eval(arg.parse_args().args)

# create folders and files
output_path = dir_path + "/../static/models/"
    + args_dict['time'].strftime("%Y%m%d%H%M%S") + "/"
os.mkdir(output_path)
os.mkdir(output_path + "images/")
f = open(output_path + "output.log", "w")
f.close()

# train
train(epochs=int(args_dict['epochs']), batch_size=int(args_dict['batch_size']),
      learning_rate=float(args_dict['learning_rate']),
      add_conv_layer=(args_dict['lenet_structure'] != 'LeNet'),
      optim=args_dict['optimizer'],
      show_loss=True, show_accuracy=True,
      utput_path=dir_path + "/../static/models/"
          + args_dict['time'].strftime("%Y%m%d%H%M%S") + "/")
os.rename(output_path + "lenet.pth",
    output_path + args_dict['time'].strftime("%Y%m%d%H%M%S") + "_lenet.pth")

# update database
updated_model = models.Record.objects.get(created_at=args_dict['time'])
updated_model.updated_at = datetime.datetime.now().replace(microsecond=0)
updated_model.save()
```

3. 结果展示界面的实现

结果展示页面从数据库中读取各个训练任务的数据，并将它们以表格的形式显示在界面上。对于这种项数不定的数据，这里采用Django框架的for标签来进行显示。而对于其中各单元格的精细控制还需要进一步借助Django模板过滤器和if标签：

```
{% for record in page %}
<tr>
    <td><a href="/model/{{ record.created_at|date:'YmdHis'}}">
      {{ record.model_name }}
```

```
    </td>
    <td>{{ record.uploader }}</td>
    <td>{{ record.created_at|date:'Y.m.d H:i:s' }}</td>
    <td>
        {% if record.created_at != record.updated_at %}
            <span style="color: green;">Complete</span>
        {% else %}
            <span style="color: blue;">Training...</span>
        {% endif %}
    </td>
    <td>
        {% if record.created_at != record.updated_at %}
            <a href="{% static 'models/' %}{{record.created_at|date:'YmdHis'}}
            /{{ record.created_at|date:'YmdHis' }}_lenet.pth"
            class="lenet" download>lenet.pth
        {% endif %}
    </td>
</tr>
{% endfor %}
```

如上，if标签在任务的发起时间与完成时间不同时，显示蓝色文本“Training...”，否则显示绿色文本“Complete”，并提供下载链接；默认的年月日格式格式较为复杂，与相应文件夹的格式不同，故采用date过滤器转化时间格式。除了提供页面上单个文件的下载链接外，此处还通过简单的JavaScript语句实现了批量下载的功能。具体而言，点击批量下载链接时，触发一个JavaScript函数download，遍历点击了页面上的所有下载链接，分别触发了下载。

```
<!-- 'download all' link -->
<th><a href="javascript:download()">Download All</th>

<!-- download function -->
<script type="text/javascript">
  function download() {
    for (var i in document.getElementsByClassName('lenet')) {
      document.getElementsByClassName('lenet')[i].click();
    }
  }
</script>
```

这里，我们通过django.core.paginator.Paginator类来实现分页功能。除了传递总页数page_num，分页器对象paginator，当前页page外，records.py还会传递一个paginator_width参数，用于在分页器显示的页码数量。在现有的参数下，通过for标签遍历页码及if条件判断，显示首尾两个固定页、当前页面及当前页面的相邻页面。修改paginator_width参数，可以改变向前向后显示相邻页面的数量。当前页面会显示为红色，提醒用户当前的页面位置。 除了中间的页码链接外，这里还在页码栏的两侧添加了指向上下页的箭头用于翻页，用if标签控制在第一页不显示前箭头，最后一页不显示后箭头。

```
<nav aria-label="paginator"><ul>
    {% if page.has_previous %}
        <a href="/records?model_name={{ filter_model_name }}
            &uploader={{ filter_uploader }}
            &page_num={{ page.previous_page_number }}">←</a>
    {% endif %}
    {% for i in paginator.page_range %}
```

```
        {% if i == 1 or i == paginator.num_pages
            or i|add:paginator_width > page.number
            and i < page.number|add:paginator_width %}
            {% if i == page.number %}
                <a href="/records?model_name={{ filter_model_name }}
                    &uploader={{ filter_uploader }}&page_num={{ i }}"
                    style="color: red;">{{ i }}</a>
            {% else %}
                <a href="/records?model_name={{ filter_model_name }}
                &uploader={{ filter_uploader }}&page_num={{ i }}">{{ i }}</a>
            {% endif %}
        {% endif %}
        {% if i > 1 and i < paginator.num_pages %}
            {% if i|add:paginator_width == page.number
                or i == page.number|add:paginator_width %}
                ..
            {% endif %}
        {% endif %}
    {% endfor %}
    {% if page.has_next %}
        <a href="/records?model_name={{ filter_model_name }}
            &uploader={{ filter_uploader }}
            &page_num={{ page.next_page_number }}">➡</a>
    {% endif %}
</ul></nav>
```

任务展示界面还实现了根据任务名称和上传者进行简单搜索的功能，提供了输入任务名称和上传者的两个文本框及一对用于选择搜索方式为“模糊”还是“精确”的单选按钮，这两种方式的逻辑分别为相应字段“包含”或“等于”输入字符串。提交搜索后，views.py中的records函数根据filter_model_name,filter_uploader和search参数对数据库内容进行筛选，并返回筛选后的结果。搜索后的分页器仍然有效。

```
# filter logic
if context['search'] == "exact":
    if model_name:
        filtered_records = filtered_records.filter(model_name=model_name)
    if uploader:
        filtered_records = filtered_records.filter(uploader=uploader)
else:
    filtered_records = filtered_records.filter
        (model_name__icontains=model_name)
    filtered_records = filtered_records.filter(uploader__icontains=uploader)
```

4. 任务详情页面的实现

任务详情页面呈现的任务基本信息及实现与任务列表页面相同，主要添加了训练日志和已完成的任务结果的可视化展示。这些资源文件事实上由launcher.py中的系统操作及其调用的train函数产生，存放于以任务创建时间命名的目录中。只需要利用从数据库中查找得到的时间戳，即可获得相应的资源文件，然后分别用<textarea>和<img>标签展示在页面上。 为了界面，且美观方便用户全选复制，这里并不把训练日志的内容直接展示在页面上，而是设置了一个只读的文本区域。对于只有在训练结束后才能得到的折线图，设置好alt属性，在训练进行时向用户展示``的提示信息。

```
<h1>Training Logs</h1>
<textarea style="border: 1px solid black; width: 25%; height: 600px;
    font-size: 15px" readonly="readonly">{{ log }}</textarea>
<h1>Image for Loss</h1>
<img src="{% static 'models/' %}{{ record.created_at|date:'YmdHis' }}
    /images/loss.png" alt="Not currently available"><br/>
<h1>Image for Accuracy</h1>
<img src="{% static 'models/' %}{{ record.created_at|date:'YmdHis' }}
    /images/accuracy.png" alt="Not currently available"><br/>
```

总结

本项目利用Django框架实现了要求中的四个界面以及作业中提及的所有必备和加分要求，并添加了我认为较为重要的关键字搜索和模型下载、批量下载的功能。具体的界面样式和用户操作方法见运行启动说明。