Background

There are **two main components** implemented.

- **UX** implemented with **JavaScript**
- **Optimization Engine** implemented with **Python**

UX Component

The **main function** of the component resides in `\afms\src\AFMS\FSOptimizer\run.js`

Description

UX will get the following inputs (`file` and `output_folder`) and post it to the **web URL** as listed below

- `file == scenarioFile` (all the parameters setting for a particular scenario)
- `output_folder == fs_optimizer_results` (output folder name)

POST TRIGGER

```
http://localhost:5000/run
```

Optimization Engine Component

The **main function** of the component resides in `\optimizer\server.py`

Description

`server.py` is the **server file** for receiving `POST` commands from the **UX component**.

The **main API call** of the component resides in `\optimizer\engine_frame.py`

> **FUNCTION**
>
> ```
> API_main_run_engine(input_folder, output_folder,
> need_pre_process=True)
> ```

- **Main API** is called by `server.py`
- *input_folder* : `scenarioFile` is extracted and placed in a folder for the optimization model
- *output_folder* : for writing out all the result files

Local Machine Setup

Let's setup the tool for usage.

Getting Started

Operating System

- Windows 10

Required Software

- Python 3.7 or higher
- SCIPOptSuite-8.0.0-win64-VS15
- Node-v16.15.0-x64

Dependencies

- Refer to `readme_6sep2022`

Running the tool locally

Front-End Setup

Client (UX Component)

1. Start **Command Prompt** or **Terminal**
2. Run the following commands

```
cd afms
yarn start
```

3. Open browser `http://localhost:8080`

Client (UX Component, no internet connectivity required)

1. Start **Command Prompt** or **Terminal**
2. Run the following commands

```
cd afms
yarn build
cd afms\dist
python -m http.server 8080
```

3. Open browser `http://localhost:8080`

Back-End Setup

Server (Optimization Engine Component)

1. Start **Anaconda Prompt**

2. Run the following commands

```
cd optimiser
```

3. Environment setup *(you only need to do this step if you are initialising the environment for the first time)*

```
conda create –n opt
conda activate opt
conda install geopandas
conda install pyproj
pip install flask
pip install flask_cors
pip install pyscipopt
```

4. Run the following commands

```
conda activate opt
python server.py
```

Docker Setup

If you wish to use the pre-built images, follow the steps below. You can either do it via Docker Compose or run two images separately.

Docker Compose

EASY SETUP

Copy the file below, save it as `docker-compose.yml`. Then run the following command in your **Command Prompt** or **Terminal**.

```
docker compose up
```

docker-compose.yml

```yaml
version: '1'
services:
  front-end:
    image: eldoraboo/afms:latest
    ports:
      - "81:80"
    depends_on:
      - back-end
  back-end:
    image: eldoraboo/optimizer:latest
    ports:
      - "5000:5000"
```

Alternatively, you may follow the instructions below to run two images separately.

Front-End Setup

```
docker pull eldoraboo/afms:latest
docker run  --name front-end -d -p 81:80 eldoraboo/afms:latest
```

Back-End Setup

```
docker pull eldoraboo/optimizer:latest
docker run --name back-end -d -p 5000:5000 eldoraboo/
optimizer:latest
```

Setting Up Docker

Setting Up Docker for the AFMS Project

Creating Dockerfiles

Front-End Dockerfile

Building Docker Images

Once you have created the Dockerfiles, you can build the Docker images.

Pushing to DockerHub

Retrieve Image ID

Exporting Docker Images

Retrieve Image Name

Pulling from DockerHub

Pulling Latest Images

Deploying Containers

Docker Run

Creating Dockerfiles

Front-End Dockerfile

Stage 1 - The Build Process

afms\Dockerfile

```
FROM node:12-alpine as build-deps
WORKDIR /usr/src/app
COPY package.json yarn.lock ./
RUN apk add --no-cache autoconf
RUN yarn
COPY . ./
RUN yarn build
```

Stage 2 - The Production Environment

afms\Dockerfile

```
FROM nginx:1.21-alpine
COPY --from=build-deps /usr/src/app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Completed Dockerfile

afms\Dockerfile

```
# Stage 1 - the build process
FROM node:12-alpine as build-deps
WORKDIR /usr/src/app
COPY package.json yarn.lock ./
RUN apk add --no-cache autoconf
RUN yarn
COPY . ./
RUN yarn build

# Stage 2 - the production environment
FROM nginx:1.21-alpine
COPY --from=build-deps /usr/src/app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Back-End Dockerfile

Stage 1 - Installing Dependencies

optimiser\Dockerfile

```
FROM continuumio/miniconda3:latest
WORKDIR /app
COPY *.py .
COPY requirements.txt .
RUN pip install -r requirements.txt
```

Stage 2 - conda & pip Setup

optimiser\Dockerfile

```
RUN pip install flask flask_cors
COPY environment.yml .
RUN conda config --set channel_priority strict
RUN conda create -n opt python=3.9
RUN /bin/bash -c "source activate opt"
RUN conda install -c conda-forge scip
RUN conda install -c conda-forge pyscipopt
RUN conda install -c conda-forge geopandas
RUN conda install -c conda-forge pyproj
```

Stage 3 - Activate Script

optimiser\Dockerfile

```
COPY server.py .
EXPOSE 5000
SHELL ["conda", "run", "-n", "opt" "/bin/bash", "-c"]
CMD ["python", "server.py"]
```

Completed Dockerfile

optimiser\Dockerfile

```
# Stage 1 - installing dependencies
FROM continuumio/miniconda3:latest
WORKDIR /app
COPY *.py .
```

Building Docker Images

Once you have created the Dockerfiles, you can build the Docker images.

Front-End Image

The `docker build` command builds the **Docker image** using the instructions in the Dockerfile. The `.` argument specifies the **location** of the Dockerfile, and the `-t` argument specifies the **name** of the image (in this case, `afms`).

```
docker build . -t afms
```

Once the Docker image has been built, you can use the following command to run the Docker container.

```
docker run -d -p 81:80 afms
```

Back-End Image

The `docker build` command builds the **Docker image** using the instructions in the Dockerfile. The `.` argument specifies the **location** of the Dockerfile, and the `-t` argument specifies the **name** of the image (in this case, `optimizer`).

```
docker build . -t optimizer
```

Once the Docker image has been built, you can use the following command to run the Docker container.

```
docker run -d -p 5000:5000 optimizer
```

Pushing to DockerHub

Retrieve Image ID

Retrieve the **image ID** using

```
docker images
```

and you should see something similar to this

```
REPOSITORY            TAG       IMAGE ID       CREATED         SIZE
eldoraboo/optimizer   latest    eddc6908c0fb   2 hours ago
4.39GB
eldoraboo/afms        latest    1d0bf74ef845   2 hours ago
44.9MB
```

Tag and Push Your Image

Tag your images with a name. In general, a good choice for a tag is something that will help you understand what this container should be used in conjunction with, or what it represents. Then push your image into DockerHub.

```
# for front-end image
docker tag 1d0bf74ef845 eldoraboo/afms:latest
docker push eldoraboo/afms

# for back-end image
```

Exporting Docker Images

Retrieve Image Name

Retrieve the **image name** using

```
docker images
```

and you should see something similar to this

```
REPOSITORY            TAG       IMAGE ID       CREATED         SIZE
eldoraboo/optimizer   latest    eddc6908c0fb   2 hours ago
4.39GB
eldoraboo/afms        latest    1d0bf74ef845   2 hours ago
44.9MB
```

Exporting as .tar files

```
# for front-end image
docker save eldoraboo/afms > afms.tar

# for back-end image
docker save eldoraboo/optimizer > optimizer.tar
```

Pulling from DockerHub

Pulling Latest Images

```
# for front-end image
docker pull eldoraboo/afms:latest

# for back-end image
docker pull eldoraboo/optimizer:latest
```

Deploying Containers

Docker Run

```
# for front-end image
docker run  --name front-end -d -p 81:80 eldoraboo/afms:latest

# for back-end image
docker run --name back-end -d -p 5000:5000 eldoraboo/
optimizer:latest
```

Docker Compose

EASY SETUP

Copy the file below, save it as `docker-compose.yml`. Then run the following command in your **Command Prompt** or **Terminal**.

```
docker compose up
```

docker-compose.yml

```
version: '1'
services:
  front-end:
    image: eldoraboo/afms:latest
```

How to Use AFMS

Features added to the existing AFMS Project.

Introduction

Widget

Load & Run Scenario

Compare Scenarios

Example Use Case

Added Features

Features added to the existing AFMS Project.

Downloading Result Files

To access the files processed in the optimizer backend, we improved the API to include a downloadfile method that allows users to **download the results** stored in the output folder…

Downloading Result Files

To access the files processed in the **optimizer** backend, we improved the **API** to include a `download_file` method that allows users to **download the results** stored in the *output folder*. Then we add a **function** in the frontend to **auto-download** the file once the **optimization process** is completed.

As only **port 80** is exposed, this method enables users to retrieve the files via the frontend.

Improving the API Function

Original `run()` function

optimiser/server.py

```
@app.route('/run', methods=["POST"])
def run():
    ...
    return jsonify({"status": "success"})
```

Improved `run()` function

optimiser/server.py

```
@app.route('/run', methods=["POST"])
```

Adding a Function to Clear Cache

We want to make sure we **do not cache the outputs** of the previous optimization runs, so we **clear the cache** after every call to the function.

optimiser/server.py

```
@app.after_request
def add_header(response):
  response.headers['X-UA-Compatible'] = 'IE=Edge, chrome=1'
  response.headers['Cache-Control'] = 'public, max-age=0'
  return response
```

Auto-Download Function

Further Improvements

Possible further improvements for the AFMS Project.

Vulnerabilities

Vulnerabilities like sonatype and CVE were detected within the images of the Docker layer.

Error Handling

For invalid input values and scenarios with no solutions, the tool does not display any message.

Vulnerabilities

Vulnerabilities like sonatype and CVE were detected within the images of the Docker layer.

- It could be fixed by altering the versions of packages used, or utilising a different function.
- Some cannot be solved by the above method, so that’s the challenge.

Error Handling

For invalid input values and scenarios with no solutions, the tool does not display any message.

- Users would not be able to know if the results are optimized.
- There is no guide on what nodes are incorrect, or what parameters must be changed to fit the scenario.

Changelog

Changelog for AFMS Project

Front-End

Back-End

Dockerfiles

Front-End

Back-End

Dockerfiles

MSO: Text Classification

Documentation for additional project done on MSO Data.

Introduction

NLI-based Zero Shot Text Classification

Sandbox

<iframe

Data Extraction

Model Setup

Complete Source Code

Introduction

NLI-based Zero Shot Text Classification

This is a method to use **pre-trained language models** to classify text **without prior training on specific labels**. They do this by treating the text to be **classified as a premise** and **creating a statement** for each possible label.

This method works really well, especially when using **big pre-trained models** like **BART** and **Roberta**.

Zero-shot Classification Pipeline

The model can be loaded with the zero-shot-classification pipeline.

```
from transformers import pipeline

classifier = pipeline("zero-shot-classification",
                      model="facebook/bart-large-mnli")
```

Classify Sequences into Class Names

You can then use this pipeline to classify sequences into any of the class names you specify. If more than one candidate label can be correct, pass multi_class=True to calculate each class independently.

```
candidate_labels = ['travel', 'cooking', 'dancing', 'exploration']
classifier(sequence_to_classify, candidate_labels,
multi_class=True)

# {'labels': ['travel', 'exploration', 'dancing', 'cooking'],
#  'scores': [0.9945111274719238,
#   0.9383890628814697,
#   0.0057061901316046715,
#   0.0018193122232332826],
#  'sequence': 'one day I will see the world'}
```

Sandbox

Data Extraction

```
import pandas as pd

def parse(file_path, sheet_name, column_name):
    df = pd.read_excel(file_path, sheet_name=sheet_name)
    column = df[column_name].tolist()
    return column

categories = list(set(parse("Copy of
parcel_sorting_feedback_2020_2022_updated.xlsx",
"parcel_sorting_feedback_2020_20", "Reporting_Category")))
feedback = parse("Copy of
parcel_sorting_feedback_2020_2022_updated.xlsx",
"parcel_sorting_feedback_2020_20", "Subject_Description")

number = len(feedback)
categories_count = {category: 0 for category in categories}
count = 0
```

Model Setup

```
from transformers import pipeline
from heapq import nlargest

classifier = pipeline("zero-shot-classification",
                      model="facebook/bart-large-mnli")

def zero_shot(doc, candidates):
    dictionary = classifier(doc, candidates)
    labels = dictionary['labels']
    scores = dictionary['scores']
    return dict(zip(labels, scores))

def top(text, cats):
    results = zero_shot(text, cats)
    topthree = nlargest(3, results, key=results.get)
    for top in topthree:
        categories_count[top] += 1
    print(f"{count}: {categories_count}")
    return topthree

for feed in feedback:
    count += 1
    top(feed, categories)

print(categories_count)
```

Complete Source Code

```
import pandas as pd

def parse(file_path, sheet_name, column_name):
    df = pd.read_excel(file_path, sheet_name=sheet_name)
    column = df[column_name].tolist()
    return column

categories = list(set(parse("Copy of
parcel_sorting_feedback_2020_2022_updated.xlsx",
"parcel_sorting_feedback_2020_20", "Reporting_Category")))
feedback = parse("Copy of
parcel_sorting_feedback_2020_2022_updated.xlsx",
"parcel_sorting_feedback_2020_20", "Subject_Description")

number = len(feedback)
categories_count = {category: 0 for category in categories}
count = 0

import concurrent.futures
from transformers import pipeline
from heapq import nlargest

classifier = pipeline("zero-shot-classification",
                      model="facebook/bart-large-mnli")

def zero_shot(doc, candidates):
    dictionary = classifier(doc, candidates)
    labels = dictionary['labels']
    scores = dictionary['scores']
    return dict(zip(labels, scores))
```