A2-README

README

Setting Up the Python Virtual Environment

To set up the Python virtual environment with all the necessary packages, follow the steps below:

```shell
python3 -m venv env
source env/bin/activate  # On Windows, use `env\Scripts\activate`
pip install -r requirements.txt
```

Special Notes

- **mlrose**: This package has been installed for part 1. It includes many optimization problems, fitness functions, and optimization algorithms. You can find the original GitHub repository here.
- **pyperch**: This package has been cloned and used for part 2. It defines neural network model classifiers that can easily switch the weight-tuning algorithms from backpropagation to optimization algorithms such as RHC, SA, and GA. The original GitHub repository is here.

Part 1: Randomized Optimization

All code related to Part 1 is included under the `part1` folder. Below is a detailed description of the structure and components:

Executable Algorithm Scripts

The following scripts implement the respective algorithms:

1. **ga.py**: Implements the Genetic Algorithm.
2. **mimic.py**: Implements the MIMIC algorithm.
3. **rhc.py**: Implements Randomized Hill Climbing.
4. **sa.py**: Implements Simulated Annealing.

Helper Files

1. **Optimization Problems:**
   Optimization problems are defined in the `problems.py` script. These problems can be called by the algorithm scripts to ensure consistency and facilitate easier cross-comparison of results.

2. **Shared Plotting Logic:**
   Shared plotting logic is included in the `utils.py` script. This script contains functions for generating plots and visualizations from the results.

3. **Configuration Files:**
   Configuration files are defined in `configs.json5`. These files contain the parameters and settings for running the optimization algorithms.

4. **Generated Data:**
   Generated data is stored in CSV format under the `optimization_results` folder. This folder also contains the generated plots and images. The results are organized by algorithm (`rhc`, `ga`, etc.) and problem type (`simple` or `complex`)

Directory Structure

```
📦part1
 ┣ 📂optimization_results
 ┃ ┣ 📂ga-TSP-complex
 ┃ ┣ 📂ga-TSP-simple
 ┃ ┣ 📂ga-knapsack-complex
 ┃ ┣ 📂ga-knapsack-simple
 ┃ ┣ 📂mimic-TSP-complex
 ┃ ┣ 📂mimic-TSP-simple
 ┃ ┣ 📂mimic-knapsack-complex
 ┃ ┣ 📂mimic-knapsack-simple
 ┃ ┣ 📂rhc-TSP-complex
 ┃ ┣ 📂rhc-TSP-simple
 ┃ ┣ 📂rhc-knapsack-complex
 ┃ ┣ 📂rhc-knapsack-simple
 ┃ ┣ 📂sa-TSP-complex
 ┃ ┣ 📂sa-TSP-simple
 ┃ ┣ 📂sa-knapsack-complex
 ┃ ┣ 📂sa-knapsack-simple
 ┣ 📜configs.json5
 ┣ 📜ga.py
 ┣ 📜mimic.py
 ┣ 📜optimize-complicated.py
 ┣ 📜optimize-simple.py
 ┣ 📜problems.py
 ┣ 📜rhc.py
```

```
├ 📜sa.py
└ 📜utils.py
```

Running the Code

To run the scripts, use the following commands:

```shell
# Part1
cd part1

# For Genetic Algorithm
python ga.py

# For MIMIC Algorithm
python mimic.py

# For Randomized Hill Climbing
python rhc.py

# For Simulated Annealing
python sa.py
```

Each script will read the problem definitions from `problems.py`, use the configurations specified in `configs.json5`, and save the results in the `optimization_results` folder.

If you want to change the hyper parameters (required for **GA** and **MIMIC** as they have two parameters to tune), you need to comment out/in the *problem solving* and *plotting* sections:

- You have to choose which parameter you want to fix and the other parameter you want to test, you can choose either the 1 or 2 in the example below
- When plotting, you have to change the `attribute_col` and `attribute` corresponding to the hyper parameter you are testing
- Same logic applies for MIMIC

```python
# For example GA
# 1. fixed population
population_sizes = [10]
mutation_rates = [0.1, 0.2, 0.3, 0.4]

# 2. fixed mutation_rates
population_sizes = range(10, 200, 50)
mutation_rates = [0.1]

# ...
plot_op_results(
    results=mimic_knapsack_complex_results,
    problem_name="knapsack-complex",
    attribute_col="Keep Percent",
    attribute="keep_percentage",
    algorithm="mimic",
    iteration_list=iteration_list,
)

plot_op_results(
    results=mimic_knapsack_complex_results,
    problem_name="knapsack-complex",
    attribute_col="Population Size",
    attribute="population",
    algorithm="mimic",
    iteration_list=iteration_list,
)
```

Data and Results

- Generated data (CSV files) and plots (images) are stored in the `optimization_results` folder.
- The results are organized into subfolders by algorithm and problem type for easy access and comparison.

Part 2: Neural Network Model Optimization

For this section, we are going to test the findings from Part 1 and apply them to Part 2 for a better understanding of the Neural Network Model. Most of the code is referenced from the Python library provided in the edDiscuss: pyperch.

The entire repo has been cloned into the codebase and imported directly. The process can be divided into three parts:

1. Load the data and do the data preprocessing.
2. Tune the best hyperparameters with gridsearch.
3. Once the best parameters are collected, use them to train, verify the NN model, and plot the results collected.

Directory Structure

```
📦part2
 ┣ 📂results
 ┃ ┣ 📜GA-accuracy.png
 ┃ ┣ 📜GA-loss.png
 ┃ ┣ 📜RHC-accuracy.png
 ┃ ┣ 📜RHC-loss.png
 ┃ ┣ 📜SA-learning-accuracy.png
 ┃ ┣ 📜SA-learning-curve.png
 ┃ ┣ 📜SA-learning-loss.png
 ┃ ┣ 📜backprop-learning-accuracy.png
 ┃ ┣ 📜backprop-learning-curve.png
 ┃ ┗ 📜backprop-learning-loss.png
 ┣ 📜backprop.py
 ┣ 📜example-backprop.py
 ┣ 📜example-ga.py
 ┣ 📜example-rhc.py
 ┣ 📜example-sa.py
 ┣ 📜ga.py
 ┣ 📜rhc.py
 ┣ 📜sa.py
```

Running the Code

To run the scripts, use the following commands:

```shell
# Part2
cd part2

# For Genetic Algorithm
python ga.py

# For Randomized Hill Climbing
python rhc.py

# For Simulated Annealing
python sa.py

# For Backpropagation
python backprop.py
```

Each script will read the data, preprocess it, perform grid search for the best hyperparameters, train the model, and save the results in the `results` folder.

Data and Results

- Generated data (CSV files) and plots (images) are stored in the `results` folder.
- The results include accuracy and loss plots for each optimization algorithm.