Assignment_2_My_First_Hillclimber_[netid]

September 7, 2021

1 Assignment 2: My First Hillclimber

In your last assignment we all got up to speed on setting up our coding environment, working through the logistics of these programming assignments, and trying to get us all on the same page for prerequisite knowledge. Now it's time to start making the backbone for our assignments (and project) for the rest of this semester, a basic evolutionary algorithm!

Starting with the most basic and vanilla starting point we can, we'll simply be implementing the Hillclimber algorithm we went over in class, and even using the same example of optimizing a bit string for the OneMax problem. So you should already know what the steps and desired outcome should be – this is just an excerise in making the code what you want! Just like real vanilla, this is itself subtlely complex and is the starting point to layer on so many extra topping, bells, and whistles to complement this base – but for this week we'll try and keep it plain and simple.

[ ]:
```python
# import required libraries

import numpy as np
import copy
import matplotlib.pyplot as plt
plt.style.use('seaborn')
```

Since we already know the pseudocode, that seems like a great place to start. Borrowing from the Luke textbook (Section 2.1, pg 17), the steps for a basic Hillclimber are as follows:

Algorithm 4 *Hill-Climbing*

```
1: $S \leftarrow$ some initial candidate solution          ▷ The Initialization Procedure
2: repeat
3:     $R \leftarrow$ Tweak(Copy($S$))                     ▷ The Modification Procedure
4:     if Quality($R$) > Quality($S$) then               ▷ The Assessment and Selection Procedures
5:         $S \leftarrow R$
6: until $S$ is the ideal solution or we have run out of time
7: return $S$
```

1.0.1 Q1: Implementation

In the spirit of starting simple, let's also begin with the most barebones implementation of this algorithm. The pseudocode below followings this proceedure (replacing the current solution, `S`, and its potential replacement, `R`, with the variable names `parent` and `child` as we used in class. Please fill in the missing code.

Hint: note that Python uses pointers, so please keep in mind the difference between and shallow vs. deep copy, if approapate for your implementation.

Hint: note also that here we're using defult parameters to pass in the hyperparameter settings for our evolutionary algorithm.

```
[ ]: def hillclimber(total_generations = 100, bit_string_length = 30,␣
      ↪num_elements_to_mutate = 1):

         # the initialization proceedure
         ...

         for ... # repeat

             # the modification procedure
             ...

             # the assessement procedure
             ...

             # selection procedure
             ...

     hillclimber() # call function to run
```

1.0.2 Q2: Record Keeping

You've now implemented an evolutionary algorithm... we think. What happens when you run it? Not much, eh? The code block above was to show you how simple the pieces of an evolutionary algorithm could be, but let's add in some record keeping to be able to observe what's going on throughout the process.

Hint: Being able to turn this output on and off with a hyperparameter is very helpful when running trials in large batches later on.

Modify the above code block such that you are able to display the generation, candidate solution, and fitness – similar to this example output:

Also modify and rerun the code, such that you are able to plot the fitness over generation time, similar to this example plot (not necessarily in exact values of shape, but in the general trend):

Hint: Don't be afraid to steal your code (or the solutions) from last assignment to build upon too.

```
[ ]: # plot fitness over time
     ...
```

1.0.3 Q3: Repeatable Experiments

In analyzing the effects of differences between different experimental setup and algorithmic variants, one of the things we'll need to do in future assignments is run, analyze, and visualize repeated trials to gain statistical signifance. Let's take a first step in that direction by leveraging the functional form of our hillcimber setup above to run and plot multiple trials.

We'll start first with running 10 trials of the same hillclimber as above, and simply overlaying the plots of each into a single figure, something like this:

```
[ ]: # plot overlayed fitness over time
     ...
```

The above rainbow of lines looks fun, but will get messy quickly. Instead for most comparisons we'll plot fitness curves with confidence intervals. My personal favorite is the bootstrapped confidence interval, as it makes very few assumptions about the distribution of your data. The downside is that it's much slower than a confidence interval based on standard deviations of a normal distribution. Please plot a 95% boostrapped confidence interval of fitness over time for 10 runs.

Hint: I like Randy's tutorial on boostrapped confidence intervals. Please note that the confidence interval values need to be calcuclated separately for each generation in a time series plot.

Hint: With a small number of runs like this, you're likely to be thrown waning from the boostrapped confidence interval funcation (which works best with a larger amount of repititions/variation). Feel free to ignore these for now (or even silence warning – though be cautious in general if surpressing all warnings).

Hint: If you're not already familiar with it, you may want to check out the fill-between and alpha function/argument in matplotlib.

It may look something like this:

```
[ ]: #plot bootstrapped fitness over time

     import warnings
     # warnings.filterwarnings('ignore') # Danger, Will Robinson! (not a scalable␣
      ↪hack, and may surpress other helpful warning other than for ill-conditioned␣
      ↪bootstrapped CI distributions)
     import scikits.bootstrap as bootstrap

     total_generations = 100
     total_runs = 10
     array_of_fitness_over_time = ...

     ...
```

1.0.4 Q4: Analysis of Results

The above outputs are one of the key ways in which we'll investigate and analyze evolutionary runs throughout the semester. What interesting trends or phenomena do you observe about the fitness-over-time plot and/or the solution output over time? What are your initial reactions to the monotonicity or convergence rate of this algorithm?

insert answer here

1.0.5 Congratulations, you made it through your first programming assignment!

Please save this file as a .ipynb, and also download it as a .pdf, uploading **both** to blackboard to complete this assignment.

For your submission, please make sure that you have renamed this file (and that the resulting pdf follows suit) to replce `[netid]` with your UVM netid. This will greatly simplify our grading pipeline, and make sure that you receive credit for your work.

Academic Integrity Attribution During this assignment I collaborated with:

list partners here