Hyperparameter Search

Friday, September 21, 2018 3:06 PM

INTRODUCTION (preaching to the choir)

- Finding good hyperparameter settings for computational models and algorithms is critical for good performance
- Rigorous computational science demands rigorous hyperparameter optimization (or sensitivity analysis in cases where outputs cannot be numerically scored)
- This is relevant for any computational analysis–not just machine learning–when we are uncertain about how to specify parameters (aka options, settings, inputs, architectures, designs, whatever terminology you prefer)
    - If the analysis has a performance score, we should find the optimal hyperparameters *and* understand the sensitivity of performance to the hyperparameters
    - If the analysis does not have a performance score, we should assess whether our interpretation of the results is sensitive to the hyperparameters
- We routinely do computational experiments/investigations of this kind, but they take much time and effort to implement and run
- Our time and effort should be spent on drug discovery research, not on hyperparameter search
- We need an easy-to-use tool that works with GSK's high performance computing systems to automatically perform distributed hyperparameter search for any computational analysis
- hp_search.py aims to accomplish this
- I hope this code will be widely used and increase the productivity of many people, making it easier to leverage large scale computing to do great science

BASICS

- **User provides a script for a given analysis**
    - Script can be written in Python, R, Perl, or conceivably any language
    - Script 1) reads hyperparameter settings and other configurations from a text file called input.json, 2) executes a single analysis, and, if the results include a performance score, 3) writes the score to a text file called output.json
- **User provides metadata for each hyperparameter to optimize or explore in model_config.json**
    - Variable type: categorical, discrete, or continuous
    - Domain: set of values or range
    - Transformation: linear or log (recommended if your range spans orders of magnitude)
- **User specifies resources required for a single analysis in job_config.json**
    - Cluster: CMS or GSKTech
    - Processor_type: CPU or GPU
    - Num_processors
    - Memory
    - Time
- **User specifies hyperparameter search settings in search_config.json**
    - Search type: line, grid, random, Bayesian Optimization (bopt), or a combination of bopt, random, and grid search
    - Time
    - Max number of hyperparameter combinations to try
    - More…
- **hp_search.py executes the hyperparameter search**
    - Jobs are distributed on the CMS or GSKTech high performance computing cluster
    - Hyperparameter settings and their corresponding performance scores are written to a text file called hp_search_data.txt

PREREQUISITES

- Access to Unix
- Access to CMS or GSKTech HPC
- Basic familiarity with Slurm
- **Read this:**
    - If you are tuning hyperparameters to optimize a performance metric, you ABSOLUTELY MUST reserve some test data that remains totally untouched until after you have made a final choice of hyperparameter values.
    - The convention is to split your data into training, validation, and test sets. The training data are used to fit the parameters of your model. The validation data are used to compare hyperparameter settings. The test data are used to report the generalization performance of your model using your final choice of hyperparameter values.
    - As you try more and more hyperparameter settings, you run the risk of finding a hyperparameter setting that happens by chance to work well on the validation data you are using to compare hyperparameter settings, but doesn't work well in general.
    - Once you are satisfied with your hyperparameter search(es) (as many as you like) and have made a final choice of hyperparameter values, then you evaluate performance on the unseen test data. If performance on the test data is worse than performance on the validation data, this is a sign you have overfit to the training+validation data.
- **Read this:**
    - If you are tuning hyperparameters to optimize a performance metric, once your hyperparameter search is complete, do NOT simply use the hyperparameter setting that performed the best.
    - It is very likely that many hyperparameter settings performed nearly as well, in which case, you should choose the hyperparameter setting that corresponds to the simplest model among the top performers.
    - You should look at the performance scores for all hyperparameter settings that were explored and assess the following:
        - What is the variance of your performance metric with fixed hyperparameter values but varying training and validation sets?
        - Given this variance, which hyperparameter settings had performance scores insignificantly different from the best performance score?
        - Among these good hyperparameter settings, chose the setting that corresponds to the simplest model (fewest parameters, strongest regularization, etc.). The simplest model is most likely to generalize the best.

GETTING STARTED

- Take a look at the other pages in this documentation
- Connect to Slurm submit node
  - CMS HPC: ushpc.gsk.com
  - GSKTech HPC: us1sxlx00202.corpnet2.com
- If on the CMS HPC, load Slurm
  ```
  $ module load slurm
  ```
- If on the GSKTech HPC, add Slurm to path
  ```
  $ export PATH=${PATH}:/usr/local/slurm/bin
  $ export MANPATH=${MANPATH}:/usr/local/slurm/share/man
  ```
- Check your python version
  ```
  $ python -V
  ```
- If NOT Python 3, add Python 3 to path
  ```
  $ export PATH=/GWD/bioinfo/projects/cb01/users/rouillard/anaconda5.1/bin:${PATH}
  ```
  (can be any Python 3; doesn't have to be my installation)
- If you use git, fork or clone the git repo
  https://git.gsk.com/andrew.d.rouillard/tutorial_hpsearch
  https://us1salx00359.corpnet2.com/andrew.d.rouillard/tutorial_hpsearch.git
  git@us1salx00359.corpnet2.com:andrew.d.rouillard/tutorial_hpsearch.git
- If you do not use git, copy the tutorial
  ```
  $ cp -r /GWD/bioinfo/projects/cb01/users/rouillard/DeepLearning/Tutorial_HPSearch
  /path/to/your/stuff/Tutorial_HPSearch
  ```
- Tutorial_HPSearch has three folders
  - input_data: A place to put data for your project(s), though your data can be anywhere you like.
  - models: A place to put code for your project(s), though your code can be anywhere you like.
  - hp_search: Contains launch_hp_search.py, hp_search.py, search_config_template.json, and job_config_template.json. Hyperparameter search results will go here.
- **Run the demo**
  - The demo will optimize hyperparameters for a variational autoencoder (VAE) trained on a subset of TCGA data (credit to Jin Yao).
  - Go to the folder in input_data for the demo project (tcga)
    ```
    $ cd /path/to/your/stuff/Tutorial_HPSearch/input_data/tcga
    ```
  - Verify the input data are there: TCGA_nanstring.csv.gz and TCGA_meta.csv.gz
  - Go to the models folder
    ```
    $ cd /path/to/your/stuff/Tutorial_HPSearch/models
    ```
  - The file train_ae.py is the script that interfaces with hp_search.py. train_ae.py will train a single VAE model.
  - The file train_ae_input_example.json is an example of the input required by train_ae.py.
  - You can run train_ae.py independent of hp_search.py. Try
    If using CMS HPC
    ```
    $ sbatch --partition=up-gpu --gres=gpu:1 --mem=32G -t 120 -o train_ae_log.txt -e train_ae_log.txt --
    wrap="/GWD/bioinfo/projects/cb01/users/rouillard/anaconda5.1/bin/python train_ae.py
    train_ae_input_example.json"
    ```
    If using GSKTech HPC
    ```
    $ sbatch --partition=us_hpc --gres=gpu:1 --mem=32G -t 120 -o train_ae_log.txt -e train_ae_log.txt --
    wrap="/GWD/bioinfo/projects/cb01/users/rouillard/anaconda5.1/bin/python train_ae.py
    train_ae_input_example.json"
    ```
    Then check if the job is running
    ```
    $ squeue -u {your mudid}
    ```
    And view progress
    ```
    $ tail -f train_ae_log.txt
    ```
    And view results when finished (should finish in about 15 minutes)
    ```
    $ cd /path/to/your/stuff/Tutorial_HPSearch/models/train_ae_results_example
    $ ls
    ```
  - Go to the folder in hp_search for the demo project (tcga)
    ```
    $ cd /path/to/your/stuff/Tutorial_HPSearch/hp_search/tcga
    ```
  - Have a look at the three files: search_config_v0.json, job_config_v0.json, and model_config_v0.json
  - Note how model_config_v0.json looks like train_ae_input_example.json, except that, for the parameters we want to optimize, instead of a single value, we have the hyperparameter search metadata specified as a dictionary of key:value pairs
  - If on the GSKTech HPC, open job_config_v0.json in a text editor and set "cluster" to "GSKTECH"
  - Go to the hp_search folder
    ```
    $ cd /path/to/your/stuff/Tutorial_HPSearch/hp_search
    ```
  - Have a look at the two files: launch_hp_search.py and hp_search.py
  - Launch the hyperparameter search
    ```
    $ python launch_hp_search.py --search_config_path tcga/search_config_v0.json --job_config_path
    tcga/job_config_v0.json --model_config_path tcga/model_config_v0.json
    ```
  - Details for this hyperparameter search are logged in tcga/launched_hp_searches.txt
    ```
    $ cat tcga/launched_hp_searches.txt
    ```
  - View progress
    ```
    $ tail -f tcga/hp_search_0/log.txt
    ```
  - View progress of individual jobs
    ```
    $ tail -f tcga/hp_search_0/hp_combination_0/log.txt
    ```
  - View hyperparameter search results
    ```
    $ cat tcga/hp_search_0/hp_search_data.txt
    ```
- **Create your own analysis script in R, Python, Perl, etc.**

    - Go to the models folder
      ```
      $ cd /path/to/your/stuff/Tutorial_HPSearch/models
      ```
    - See my_analysis_template.py and my_analysis_template.R for template scripts in Python and R, respectively.
    - Add the input variables to your analysis to my_input_template.json. Can use arbitrary placeholder values.
    - Make sure you can run a single instance of your analysis before moving on to hyperparameter search.
    - Congrats, you have completed 95% of the work needed to run the hyperparameter search!
- **Create your own hyperparameter search**
    - Go to the hp_search folder
      ```
      $ cd /path/to/your/stuff/Tutorial_HPSearch/hp_search
      ```
    - Create a project folder
      ```
      $ mkdir my_project
      ```
    - Copy search_config_template.json and job_config_template.json into your project folder
      ```
      $ cp search_config_template.json my_project/search_config_v0.json
      $ cp job_config_template.json my_project/job_config_v0.json
      ```
    - Customize search_config_v0.json and job_config_v0.json as desired
    - Copy my_input_template.json into your project folder, renaming to model_config.json
      ```
      $ cp /path/to/your/stuff/Tutorial_HPSearch/models/my_input_template.json my_project/model_config_v0.json
      ```
    - Customize model_config_v0.json as desired, plugging in hyperparameter search metadata for the input variables you want to tune/explore
    - Launch the hyperparameter search
      ```
      $ python launch_hp_search.py --search_config_path my_project/search_config_v0.json --job_config_path
      my_project/job_config_v0.json --model_config_path my_project/model_config_v0.json
      ```
    - Details for this hyperparameter search are logged in my_project/launched_hp_searches.txt
      ```
      $ cat my_project/launched_hp_searches.txt
      ```
    - View progress
      ```
      $ tail -f my_project/hp_search_0/log.txt
      ```
    - View progress of individual jobs
      ```
      $ tail -f my_project/hp_search_0/hp_combination_0/log.txt
      ```
    - View hyperparameter search results
      ```
      cat my_project/hp_search_0/hp_search_data.txt
      ```

DISCLAIMERS

- This code needs more testing! There may be bugs! Please let me know if you find any!
- The demo makes use of code for training VAEs (Tutorial_HPSearch/models/autoencoders.py) written Summer 2018 by Olivia Lang, one of our undergraduate student interns. Olivia is very talented and I trust she got the implementation correct, but there may be bugs. Again, let me know if you find any!
- If you choose to run your hyperparameter search with Bayesian Optimization, be aware that Bayesian Optimization is an active area of research and may not work as expected on your problem. This is why I have made it possible to hedge in several ways, such as running hyperparameter search with a mixture of grid search, random search, and Bayesian Optimization. I would greatly appreciate any feedback you can provide on whether bopt is making smart exploitation/exploration suggestions or exhibiting any sort of pathological behavior.

FUTURE

- I hope this code will be widely used and increase the productivity of many people, making it easier to leverage large scale computing to do great science.
- I would greatly appreciate help of any kind
    - Testing the code and providing feedback
    - Providing more demos
    - Providing code for downstream analysis of the hyperparameter search results (diagnostics, visualizations, etc.)
    - Improving the code
    - Improving the documentation
    - "Productionizing"
    - Figuring out how to run this on RDIP
    - ...
- A couple of areas for methodological improvement
    - Automatic tuning of HMC when using MCMC to infer the GP parameters, and diagnostics to assess whether HMC sampler has converged (Is MCMC even worthwhile?)
    - Explore if there are better packages for Bayesian Optimization than GPyOpt
    - Frame hyperparameter search as an inference problem, get an ensemble of hyperparameter combinations, and average over this ensemble instead of picking a single hyperparameter combination

launch_hp_search.py

Monday, September 24, 2018 12:54 PM

Usage

```
$ python launch_hp_search.py --search_config_path my_project_path/my_search_config.json --job_config_path
my_project_path/my_job_config.json --model_config_path my_project_path/my_model_config.json
```

Inputs

- search_config_path: path to .json file with configurations for hyperparameter search
- job_config_path: path to .json file with resource requirements for individual jobs
- model_config_path: path to .json file with hyperparameter metadata and other settings for your model

What it does

- Creates folders and log files
- Submits your hyperparameter search to the Slurm queue with the Slurm settings specified in 'my_search_config.json'

Outputs

- A master job that will coordinate the hyperparameter search is submitted to the Slurm queue
- The slurm job_id and other metadata are recorded in 'my_project_path/launched_hp_searches.txt'
    - job_id: job_id assigned by Slurm
    - project_folder: '{my_project_path}', i.e. folder where my_job_config.json is located
    - search_folder: '{project_folder}/hp_search_{search_id}'
    - search_id: automatically incremented every time a hyperparameter search is launched from the same project_folder. Allows mult iple hyperparameter searches to be run in parallel.
    - log_path: '{search_folder}/log.txt'
    - search_config_path: same as input
    - job_config_path: same as input
    - search_command: command that Slurm will execute to run the master job for hyperparameter search, e.g. 'python search_hp.py --search_config_path {search_config_path} --job_config_path {job_config_path}'
    - slurm_command: unix command that will submit master job for hyperparameter search to the Slurm queue, e.g. 'sbatch -n 1 -c 4 --mem=32G -t 720 -o {log_path} -e {log_path} --wrap="{search_command}"'

What you need to know

- How to configure hyperparameter search - see search_config.json page
- How to configure individual jobs - see job_config.json page
- How to format hyperparameter metadata - see model_config.json page
- log_path for monitoring progress: '{project_folder}/hp_search_{search_id}/log.txt'
- search_folder for viewing results: '{project_folder}/hp_search_{search_id}'

hp_search.py

Tuesday, September 25, 2018 2:59 PM

Usage

```
$ python hp_search.py --search_config_path my_project_path/my_search_config.json --job_config_path
my_project_path/my_job_config.json --model_config_path my_project_path/my_model_config.json --search_id
search_id
```

Note, in practice, you won't call hp_search.py directly. It will be called by launch_hp_search.py.

Inputs

- search_config_path: path to .json file with configurations for hyperparameter search
- job_config_path: path to .json file with resource requirements for individual jobs
- model_config_path: path to .json file with hyperparameter metadata and other settings for your model
- search_id: ID number for the hyperparameter search

What it does

- Parses 'my_model_config.json' to identify which hyperparameters you want to explore and the range or set of values you want to explore for each hyperparameter
- Generates hyperparameter combinations according to your specifications in my_search_config.json, performing line search, grid search, random search, Bayesian Optimization (bopt), or a combination of bopt, random, and grid search
- Creates folders, log files, and input files to run your analysis and collect results for each hyperparameter combination
- Coordinates parallel execution of jobs with the different hyperparameter combinations on GSK's high performance computing cluster
- Monitors progress of individual jobs and collects results

Outputs

- Jobs running your model/algorithm/analysis with different hyperparameter combinations are submitted to the Slurm queue
  - Each hyperparameter combination is assigned a combination_id
- Files for a given hyperparameter combination are saved in 'my_project_path/hp_search_{search_id}/hp_combination_{combination_id}'
  - input.json
  - log.txt
  - output.json (if created by your script)
  - Other outputs from your script (assuming it uses relative paths)
- Metadata and results (if applicable) for all hyperparameter combinations are saved in 'my_project_path/hp_search_{search_id}/ hp_search_data.txt'. Each row corresponds to a hyperparameter combination. Columns are:
  - combination_id: assigned by hp_search.py
  - job_id: assigned by Slurm
  - job_time: approximate running time of job in minutes
  - previous_results_count: number of results collected at time of job submission
  - search_type: 'line', 'grid', 'random', or 'bopt' (will actually appear as '{kernel_name}_{acquisition_type}', e.g. 'Matern52_EI_MCMC')
  - is_queued: boolean indicating whether or not job was submitted to Slurm
  - model_config_path: 'my_project_path/hp_search_{search_id}/hp_combination_{combination_id}/input.json'
  - loss: model error (if applicable)
  - hyperparameters: remaining columns

What you need to know

- How to configure hyperparameter search - see search_config.json page
- How to configure individual jobs - see job_config.json page
- How to format hyperparameter metadata - see model_config.json page
- log_path for monitoring progress of master job coordinating hyperparameter search: 'my_project_path/hp_search_{search_id}/log.txt'
- log_path for monitoring progress of a job running a given hyperparameter combination: ''my_project_path/hp_search_{search_id}/hp_combination_{combination_id}/log.txt'
- search_folder for viewing results: 'my_project_path/hp_search_{search_id}'

my_analysis.script

Tuesday, October 02, 2018 1:21 PM

Usage

```
$ path/to/environment path/to/my_analysis.script path/to/input.json
```

Examples

```
$ /GWD/bioinfo/projects/RD-TSci-Software/CB/linuxbrew/bin/Rscript
/GWD/bioinfo/projects/cb01/users/rouillard/DeepLearning/Tutorial_HPSearch/models/my_analysis_
template.R path/to/input.json

$ /GWD/bioinfo/projects/cb01/users/rouillard/anaconda5.1/bin/python
/GWD/bioinfo/projects/cb01/users/rouillard/DeepLearning/Tutorial_HPSearch/models/train_ae.py
/GWD/bioinfo/projects/cb01/users/rouillard/DeepLearning/Tutorial_HPSearch/models/train_ae_inp
ut_example.json

$ /GWD/bioinfo/projects/cb01/users/rouillard/anaconda5.1/bin/python
/GWD/bioinfo/projects/cb01/users/rouillard/DeepLearning/Tutorial_HPSearch/models/my_analysis_
template.py path/to/input.json
```

Note, in practice, you won't call my_analysis.script directly. It will be called by hp_search.py.

Inputs

- Path to a json-formatted dictionary that specifies all the variables needed for your analysis.

What it does

- Reads/Parses the dictionary in the input json file.
- Executes your analysis using the variable settings specified in the input json file.
- [Optional] Writes a performance metric for your analysis to an output json file.
- [Optional] Saves other files from your analysis.
- [Optional] Anything else you want it to do.

Outputs

- [Optional] A performance metric for your analysis is saved as '{save_folder}/output.json'
  - hp_search.py automatically sets save_folder = 'my_project_path/hp_search_{search_id}/hp_combination_{combination_id}'
- [Optional] Other outputs from your analysis can be saved in save_folder or wherever you like.

What you need to know

- This is a script that runs a single instance of your analysis.
- The script can be written in Python, R, Perl, or conceivably any language.
- The script can contain the code for your entire analysis or simply be a wrapper that enables other script(s) to interface with hp_search.py.
- To interface with hp_search.py, the script must accept a single input: a path to a json-formatted dictionary that specifies all the variables needed for your analysis.
  - hp_search.py automatically specifies the path '{save_folder}/input.json' when it calls your script.
  - save_folder = 'my_project_path/hp_search_{search_id}/hp_combination_{combination_id}'
  - **model_config.json must contain the same variables that your script expects in the input json. hp_search.py uses model_config.json as a template for creating the input json.**
- The script must be able to read/parse the dictionary in the json file.
- From this point, you can use the variables in the dictionary and/or pass them to other functions or scripts.
- If your analysis computes a performance metric that you would like hp_search.py to access, your script must save this to '{save_folder}/output.json'.
  - hp_search.py automatically specifies save_folder in input.json.
  - output.json can contain a single value, list of values, or dictionary of key:value pairs
    - If list of values, hp_search.py uses the first value
    - If dictionary of key:value pairs, hp_search.py uses the value for the key 'objective'
- See train_ae.py for an example script

- See my_analysis_template.py and my_analysis_template.R for template scripts in Python and R, respectively

search_config.json

Tuesday, September 25, 2018 5:01 PM

There are many search configuration variables, but in most cases you should only need to worry about the few marked "yes" in the "you_must_decide" column.
See Tutorial_HPSearch/hp_search/search_config_template.json

search_config_variable	type	values	recommendation	you_must_decide	description	relevance
search_type	str	line, grid, random, bopt	bopt	yes	Line search: Use line search to vary one hyperparameter at a time. A base hyperparameter combination is defined by taking the middle value of each hyperparameter. Then new combinations are generated by varying one hyperparameter at a time, while all other hyperparameters are fixed to the base values. Grid search: Use grid search to generate regularly spaced hyperparameter combinations within the hyperparameter space you have specified. Random search: Use random search to generate randomly spaced hyperparameter combinations within the hyperparameter space you have specified. Bayesian Optimization (bopt): Use bopt to iteratively estimate the next best hyperparameter combination given results for previously explored hyperparameter combinations. Bopt also allows you to do a mixture of bopt, grid, and random search if grid_suggestion_probability > 0 and/or random_suggestion_probability > 0.	
search_time	float	(0, ?)	24	yes	Total time (hours) allocated for hyperparameter search. NOT the time needed to run your analysis on a single hyperparameter combination. See job_config.json to set the time for a single run of your analysis.	Mainly relevant for search_type = random or bopt. If you are doing line or grid search, make sure search_time won't terminate the hyperparameter search before all combinations have been submitted.
search_memory	int	[1, ?)	32	no	Total memory (GB) allocated for hyperparameter search. Unrelated to memory needed to run your analysis. See job_config.json to set the memory for a single run of your analysis.	
search_environment_path	str	/GWD/bioinfo/projects/cb01/users/rouillard/anaconda5.1/bin/python	/GWD/bioinfo/projects/cb01/users/rouillard/anaconda5.1/bin/python	no	Path to installation of Python 3 needed to run hyperparameter search. Can be any installation of Python 3 with the requisite packages, notably GPy and GPyOpt. This is unrelated to the environment (Python, R, Perl, etc.) needed to run your analysis. See job_config.json to set the environment for your analysis.	
search_script_path	str	hp_search.py	hp_search.py	no	Path to hyperparameter search script. Makes it easy to swap in different versions of hp_search.py if you want to work on improving the code.	
max_suggestions	int	[1, inf)	200	yes	Maximum number of hyperparameter combinations to try.	Mainly relevant for search_type = random or bopt. If you are doing line or grid search, make sure max_suggestions is greater than the number of hyperparameter combinations (determined by number of hyperparameters and num_grid_points_per_hyperparameter).
max_active_points	int	[1, inf)	20	no	Maximum number of hyperparameter combinations to run in parallel. The CMS cluster has about 220 GPUs. If you set max_active_points to 220 or higher, Slurm will automatically throttle your usage of the cluster, but you should be considerate and self-regulate to say 10% or 20% of the GPUs. Likewise for CPUs. The CMS cluster has about 3000 CPUs.	
num_grid_points_per_hyperparameter	int	[0, inf)	3	yes	Number of values for each hyperparameter if doing line or grid search. Choice depends on number of hyperparameters, time to run each job, and how long you're willing to wait for the results. Includes upper and lower bounds, so for example, num_grid_points_per_hyperparameter = 3 will yield grid = [lower bound, midpoint, upper bound] for each hyperparameter. For categorical variables, num_grid_points_per_hyperparameter is ignored and all values you specified are used. For discrete variables, the actual number of values for a hyperparameter is min(num_grid_points_per_hyperparameter, number of values you specified).	Only relevant for search_type = line, grid, or bopt (with grid_suggestion_probability > 0).
min_initial_points	int	[1, inf)	5	no	Number of results for previously explored hyperparameter combinations before bopt starts making suggestions. Choice depends on the number of hyperparameters - more hyperparameters means more points are needed before you get sensible bopt suggestions. A random combination is suggested when min_initial_points is not satisfied.	Only relevant for search_type = bopt.
y_transformation	str	none, log, neg, or neglog	log	yes	Transformation to apply to your model's objective function values (if applicable). Bopt assumes it is fitting the loss or error of your model, where smaller values indicate better performance. Log: y = log10(your model's output). Consider using log if your model's loss function values vary over orders of magnitude. Log-transformation may improve the Gaussian Process fits, yielding better bopt suggestions. Neg: y = -(your model's output). Use neg if your model outputs performance scores or goodness-of-fit values, such as log-likelihoods. Neglog: y = -log10(your model's output). Consider using neglog if your model outputs performance scores or goodness-of-fit values that vary over orders of magnitude, such as likelihoods.	Only relevant for search_type = bopt.
y_failure_value	str or float	(inf, inf), nan, max	max	no	Loss value to impute if your analysis for a given hyperparameter combination crashes or fails to finish in the time you allocated. Max: regularly update loss value for failed hyperparameter combinations to 1.05*(largest loss seen so far). This is a way of letting bopt know these are bad hyperparameter combinations. Nan: Bopt will ignore these hyperparameter combinations when suggesting new hyperparameter combinations.	Only relevant for search_type = bopt.
grid_suggestion_probability	float	[0, 1]	0	no	Probability of suggesting a hyperparameter combination from the set of grid points at each iteration.	Only relevant for search_type = bopt.
random_suggestion_probability	float	[0, 1]	0.1	no	Probability of suggesting a random hyperparameter combination at each iteration. (Actually, this is a conditional probability, given a grid combination is not suggested.)	Only relevant for search_type = bopt.
bopt_config.acquisition_type	str or list	EI, MPI, LCB, [EI, MPI], [EI, LCB], [MPI, LCB], [EI, MPI, LCB]	[EI, MPI, LCB]	no	Acquisition function bopt uses to suggest a hyperparameter combination. EI: Expected Improvement, MPI: Maximum Probability of Improvement, LCB: Lower Cost Bound. These are different ways of finding the best trade-off between exploitation (suggesting a hyperparameter combination	Only relevant for search_type = bopt.

					close to known good hyperparameter combinations) and exploration (suggesting a hyperparameter combination in an unexplored region of the hyperparameter space). Usually, the three acquisition functions perform similarly, but sometimes one will fail. Rather than picking a single acquisition function and hoping for the best, you can choose all three acquisition functions, and hp_search will randomly choose one of the three at each iteration.	
bopt_config.acquisition_optimizer_type	str	lbfgs	lbfgs	no	Optimizer bopt uses to find optimum of acquisition function.	Only relevant for search_type = bopt.
bopt_config.model_type	str	GP	GP	no	Model bopt uses to estimate loss as a function of hyperparameters. GP: Gaussian Process.	Only relevant for search_type = bopt.
bopt_config.model_estimation_method	str or list	MLE, MCMC, [MLE, MCMC]	[MLE, MCMC]	no	Method bopt uses to estimate the parameters of the Gaussian Process model given a set of hyperparameter combinations and their loss values (i.e. the training examples). MLE: Maximum Likelihood Estimation, fast but may overfit, especially with few training examples relative to number of GP parameters. (Number of GP parameters = number of hyperparameters + 2 or 3, depending on the kernel function.) MCMC: Markov Chain Monte Carlo (specifically, Hamiltonian Monte Carlo - HMC), slow but will not overfit, but may be biased if priors are mis-specified or HMC sampler is not properly tuned. If MCMC is done properly, it should outperform MLE, but it is costly to tune the HMC sampler. (HMC tuning can be automated, but this is not currently supported.) Rather than picking a single model estimation method and hoping for the best, you can choose both MLE and MCMC, and hp_search will randomly choose one of the two at each iteration. The probabilities of choosing MLE or MCMC will be automatically adjusted to favor MLE later on when there are many training examples.	Only relevant for search_type = bopt.
bopt_config.kernel	str or list	ExpQuad, RatQuad, Matern52, [ExpQuad, RatQuad], [ExpQuad, Matern52], [RatQuad, Matern52], [ExpQuad, RatQuad, Matern52]	[ExpQuad, RatQuad, Matern52]	no	Kernel (aka covariance) function bopt uses for the Gaussian Process. ExpQuad: Exponentiated Quadratic, aka Gaussian, aka Radial Basis Function. RatQuad: Rational Quadratic. Matern52: Matern 5/2. These are all stationary kernel functions that allow fits with different degrees of smoothness between training examples. Rather than picking a single kernel function and hoping for the best, you can choose all three kernel functions, and hp_search will randomly choose one of the three at each iteration. More kernels are available if you want to experiment.	Only relevant for search_type = bopt.
bopt_config.normalize_Y	bool	true, false	TRUE	no	Option for bopt to normalize the loss values before fitting the Gaussian Process.	Only relevant for search_type = bopt.
bopt_config.de_duplication	bool	true, false	TRUE	no	Option for bopt to avoid suggesting hyperparameter combinations that are already queued or in progress.	Only relevant for search_type = bopt.
bopt_config.exact_feval	bool	true, false	FALSE	no	Option for bopt to treat loss values as exact. Set to TRUE if your analysis produces exactly the same result for every run with the same hyperparameter combination.	Only relevant for search_type = bopt.
bopt_config.eps	float	[0, inf)	0	no	Minimum distance a new bopt hyperparameter combination must be from previous hyperparameter combinations. It is impractical to set this to anything other than 0 because it is unclear how distance is measured.	Only relevant for search_type = bopt.
bopt_config.n_burnin	int	[1, inf)	500	no	Number of samples to allocate for equilibration phase of HMC sampler. 500 samples takes ~1 minute with ~10 training examples and ~10 minutes with ~100 training examples.	Only relevant for search_type = bopt.
bopt_config.n_samples	int	[1, inf)	50	no	Number of samples for HMC sampler to collect after equilibration. These samples approximate the posterior distribution of the parameters of the Gaussian Process.	Only relevant for search_type = bopt.
bopt_config.step_size	float	[0, inf)	0.1	no	Step size for HMC sampler.	Only relevant for search_type = bopt.
bopt_config.subsample_interval	int	[1, inf)	10	no	Subsampling interval for HMC sampler. If n_samples is 50 and subsample_interval is 10, then the HMC sampler actually collects 500 samples (after equilibration) and keeps every 10th. This provides some protection against a small step_size leading to highly auto-correlated samples.	Only relevant for search_type = bopt.
bopt_config.leapfrog_steps	int	[1, inf)	20	no	Number of steps of the Hamiltonian dynamics simulation for each HMC sample.	Only relevant for search_type = bopt.

job_config.json

Tuesday, October 02, 2018 12:22 PM

The job configuration variable settings should be customized for your analysis. These variables specify the requirements for a single run of your analysis.
See Tutorial_HPSearch/hp_search/job_config_template.json

job_config_variable	type	values	example	you_must_decide	description
environment_path	str	Rscript, python, perl, others?	/GWD/bioinfo/projects/cb01/users/rouillard/anaconda5.1/bin/python	yes	Path to environment (Python, R, Perl, etc.) needed to run your analysis.
script_path	str		../models/train_ae.py	yes	Path to script that runs a single analysis.
cluster	str	CMS, GSKTECH	CMS	yes	High performance computing cluster where jobs will be submitted.
time	float	(0, ?)	2	yes	Time (hours) allocated for a single run of your analysis.
memory	int	[1, ?)	32	yes	Memory (GB) allocated for a single run of your analysis.
processor_type	str	CPU, GPU	GPU	yes	Type of processor on which to run your analysis.
num_processors	int	[1, ?)	1	yes	Number of processors allocated for a single run of your analysis.

model_config.json

Tuesday, October 02, 2018 12:28 PM

- Both the variables and their settings in model_config.json should be customized for your analysis.
- All inputs to your analysis must be specified here. That is, **model_config.json must contain the same variables that your my_analysis.script expects in its input json.** hp_search.py uses model_config.json as a template for creating the input json. (See my_analysis.script page.)
- There are no other required variables.
- Include both constant/fixed variables and variables you want to treat as hyperparameters.
- hp_search.py will automatically recognize which variables you want to treat as hyperparameters because you will provide the hyperparameter metadata in a dictionary as follows
  - Hyperparameter metadata are provided using dictionaries with the following fields:

field	value
type	continuous, discrete, or categorical
domain	If type is continuous, domain must be specified as a range: [lower_bound, upper_bound]. If type is discrete or categorical, domain must be specified as a list of values.
transformation	linear: hp_search.py will shift and scale the variable to range from 0 to 1. log: hp_search.py will log10-transform the variable and then shift and scale it to range from 0 to 1. Consider using log if the variable's domain spans orders of magnitude. Log-transformation may improve the Gaussian Process fits, yielding better bopt suggestions. If not using bopt, you still may prefer grid or random search values to be uniformly distributed on a log scale. NOTE, hp_search.py uses the transformed variables to generate hyperparameter combinations, then inverts the transformations before submitting jobs to run your analysis. Also, transformation is ignored for categorical variables and need not be provided.

  - Examples:
    - {"type" : "continuous", "domain" : [10, 10000], "transformation" : "log"}
    - {"type" : "discrete", "domain" : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], "transformation" : "linear"}
    - {"type" : "categorical", "domain" : ["tanh", "relu"]}
- The example below is for using train_ae.py to train a variational autoencoder. See Tutorial_HPSearch/hp_search/tcga/model_config_v0.json and Tutorial_HPSearch/models/train_ae.py.
- See also my_analysis_template.py, my_analysis_template.R, and my_input_template.json in Tutorial_HPSearch/models for generic templates.

model_config_variable	type	values	example	you_must_decide	description
save_folder	str		.	no	Folder where output.json will be saved. output.json reports the validation error of the model. save_folder is automatically specified by hp_search.py, so it does not need to be specified here.
data_file	str		../input_data/tcga/TCGA_nanostring.csv.gz	yes	Path to input data matrix. CSV or CSV.GZ file format. Rows correspond to samples and columns correspond to features. First row contains the column/feature labels. First column contains unique identifiers for the rows/samples.
meta_file	str		../input_data/tcga/TCGA_meta.csv.gz	yes	Path to meta data matrix. CSV or CSV.GZ file format. Rows correspond to samples and columns correspond to metadata. First rowcontains the column/metadata labels. First column contains unique identifiers for the rows/samples.
init_h_dim	int	[1, inf)	{"type" : "continuous", "domain" : [10, 10000], "transformation" : "log"}	no	Number of dimensions/neurons in the first hidden layer of the autoencoder.
latent_dim	int	[1, inf)	2	yes	Number of dimensions/neurons in the bottleneck/embedding/latent layer of the autoencoder.
num_layers	int	[1, inf)	{"type" : "discrete", "domain" : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], "transformation" : "linear"}	no	Number of hidden layers, not including the latent layer, i.e. number of hidden layers between the input and latent layers.
dim_scaling	str	linear, log	"log"	no	Heuristic used to set sizes of hidden layers between the first hidden layer (size determined by init_h_dim) and the bottleneck layer (size determined by latent_dim). Linear: hidden layer size is proportional to hidden layer depth, i.e. hidden layer size decreaseslinearly from init_h_dim to latent_dim. Log: log of hidden layer size is proportional to hidden layer depth, i.e. hidden layer size decreases exponentially from init_h_dim to latent_dim.
batch_fraction	int	(0, 1]	{"type" : "continuous", "domain" : [0.001, 1], "transformation" : "log"}	yes	Fraction of training examples to use at each iteration of stochastic gradient descent. Actual batch size = max(1, batch_fraction*num_training_examples). A lower bound of 10/num_training_examples should be reasonable.
learning_rate	float	(0, inf)	{"type" : "continuous", "domain" : [0.000001, 10.0], "transformation" : "log"}	no	Initial guess for the step size during stochastic gradient descent. ADAM optimizer adapts the step size as training proceeds.
epochs	int	[1, inf)	10000	no	Number of passes over the training data during stochastic gradient descent. 1 epoch = (num training examples)/(batch_size) training steps. If batch_size = 10 and there are 1000 training examples, then 100 training steps = 1 epoch. Set to a very large number because we are monitoring for overfitting want the point where overfitting begins to be the stopping point.
minutes	float	(0, inf)	60	yes	Time (minutes) allocated for training. Use this time limit to ensure the analysis exits properly, instead of being killed bySlurm when job_config['time'] is exceeded.
activation	str	sigmoid, tanh, relu, softmax, elu, selu, softplus, softsign, hard_sigmoid, linear	{"type" : "categorical", "domain" : ["tanh", "relu"], "transformation" : "none"}	no	Activation function to apply to neurons/hidden units. Tanh (or sigmoid) and relu are most common.
data_split	float	(0, 1)	0.2	no	Fraction of rows/samples in data_file to reserve for validation. Note, you MUST also reserve a fraction of samples for finaltesting. These test samples should not be in data_file.
model_type	str	Gaussian, Bernoulli	Gaussian	yes	Determines how your input data are standardized and how reconstruction errors are computed. Bernoulli: standardizes featuresto range from 0 to 1, applies a sigmoid activation to the reconstruction layer, and uses cross-entropy loss for the reconstruction error. Gaussian: z-scores features, applies a linear activation to the reconstruction layer, and uses squared error loss for the reconstructionerror.
ae_type	str	VAE	VAE	no	Type of autoencoder. VAE: variational autoencoder.
keep_weights	bool	true, false	FALSE	no	Whether or not to save the model parameters. These models can be large, and if you run many hyperparameter combinations, thiswill eat up a ton of storage.