**Initial codebase**

Describe how you identified the codebase used for your initial query, noting the source if it was pulled from a publicly available library or based on something you have previously developed. Summarise your decision-making on why you choose this code as a starting point.

**Answer**:

I used bayesian-optimization library to construct the initial codebase. This is a constrained global optimization package built upon bayesian inference and gaussian process, that attempts to find the maximum value of an unknown function in as few iterations as possible. This technique is particularly suited for optimization of high cost functions, situations where the balance between exploration and exploitation is important.

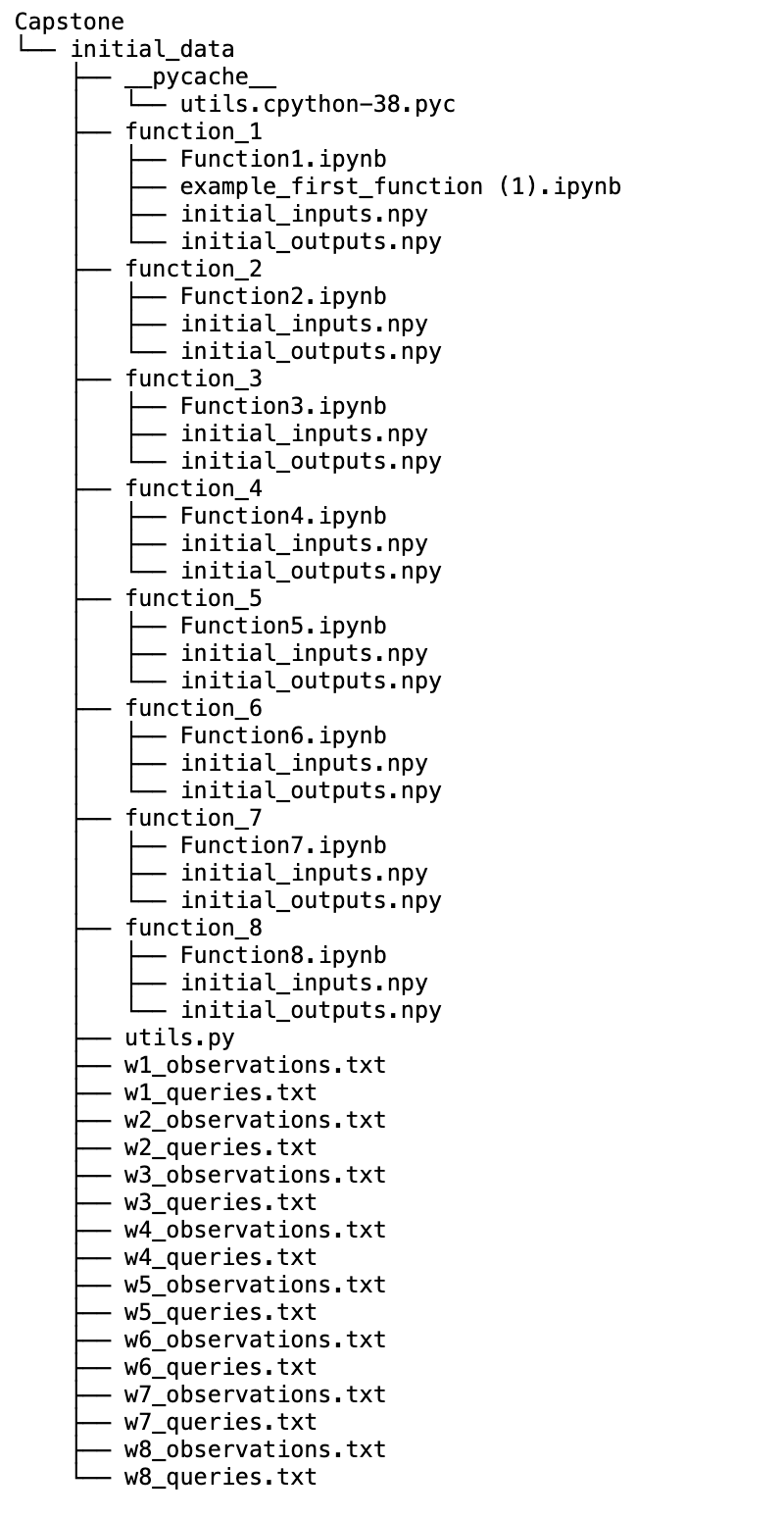
Bayesian optimization works by constructing a posterior distribution of functions (gaussian process) that best describes the function you want to optimize. As the number of observations grows, the posterior distribution improves, and the algorithm becomes more certain of which regions in parameter space are worth exploring and which are not, as seen in the picture below.

As you iterate over and over, the algorithm balances its needs of exploration and exploitation taking into account what it knows about the target function. At each step a Gaussian Process is fitted to the known samples (points previously explored), and the posterior distribution, combined with a exploration strategy (such as UCB (Upper Confidence Bound), or EI (Expected Improvement)), are used to determine the next point that should be explored (see the gif below).

This process is designed to minimize the number of steps required to find a combination of parameters that are close to the optimal combination. To do so, this method uses a proxy optimization problem (finding the maximum of the acquisition function) that, albeit still a hard problem, is cheaper (in the computational sense) and common tools can be employed. Therefore Bayesian Optimization is most adequate for situations where sampling the function to be optimized is a very expensive endeavor.

The directory setup is as follows:

For each function (1-8), there is a specific jupyter notebook and all weekly observations are combined iteratively with initial data.



The approach is as follows: (For example for function 3)

To solve this drug discovery problem using Bayesian Optimization with the hint that one of the variables (compounds) may not have any effect, you would approach it as follows:

**1. Define the Optimization Problem**

Your goal is to minimize the adverse reaction to a drug, which is quantified by a measurement that should be as close to zero as possible. You're combining three compounds, so your input space is three-dimensional. The hint suggests that one of these compounds may be inert, meaning it does not contribute to the adverse reaction.

**2. Set Up the Bayesian Optimization Framework**

You'll need a Bayesian Optimization library like bayesian-optimization. If not already installed, you can install it via pip:

pip install bayesian-optimization

**3. Define the Objective Function**

This is the function that Bayesian Optimization will minimize. It should take the amounts (or concentrations) of the three compounds as inputs and return the measured adverse reaction.

**def** drug\_reaction(compound1, compound2, compound3):

*# This function should return the adverse reaction measurement for the given combination of compounds.*

*# The measurement should be obtained from your experimental data or simulation.*

**pass**

**4. Initialize the Optimizer**

Define the bounds for each compound, then initialize the Bayesian Optimizer.

**from** **bayes\_opt** **import** BayesianOptimization

*# Define bounds for each compound (example: 0 to 1)*

pbounds = {'compound1': (0, 1), 'compound2': (0, 1), 'compound3': (0, 1)}

*# Initialize Bayesian Optimization*

optimizer = BayesianOptimization(f=drug\_reaction, pbounds=pbounds, random\_state=1)

**5. Incorporate Initial Data**

If you have initial data (previously tested combinations of compounds and their reactions), register this data with the optimizer. This step is crucial for informing the optimizer about the problem space.

*# Assuming X\_initial is your initial compounds data and Y\_initial is the corresponding adverse reactions*

**for** i **in** range(len(X\_initial)):

optimizer.register(params={'compound1': X\_initial[i][0], 'compound2': X\_initial[i][1], 'compound3': X\_initial[i][2]}, target=Y\_initial[i])

**6. Run the Optimization**

Start the optimization process, where the algorithm will suggest new combinations of compounds to test.

optimizer.maximize(init\_points=2, n\_iter=10) *# You can adjust these numbers based on your requirements*

**7. Analyze the Results**

After the optimization, analyze the results to find the combination of compounds that minimizes the adverse reaction. Pay attention to the inert compound hinted at; it might show up as having a negligible effect on the outcome.

**8. Considerations and Next Steps**

* **Experimentation**: The optimizer suggests combinations to test. You'll need to conduct experiments or simulations for each suggestion to get the adverse reaction measurement.
* **Hint Utilization**: Consider analyzing the results specifically to identify the inert compound. If you can confidently identify it, you might simplify your problem to a two-dimensional optimization (just the two active compounds).
* **Refinement and Validation**: After finding a promising combination, further experiments or more refined optimization might be necessary to validate and improve the results.

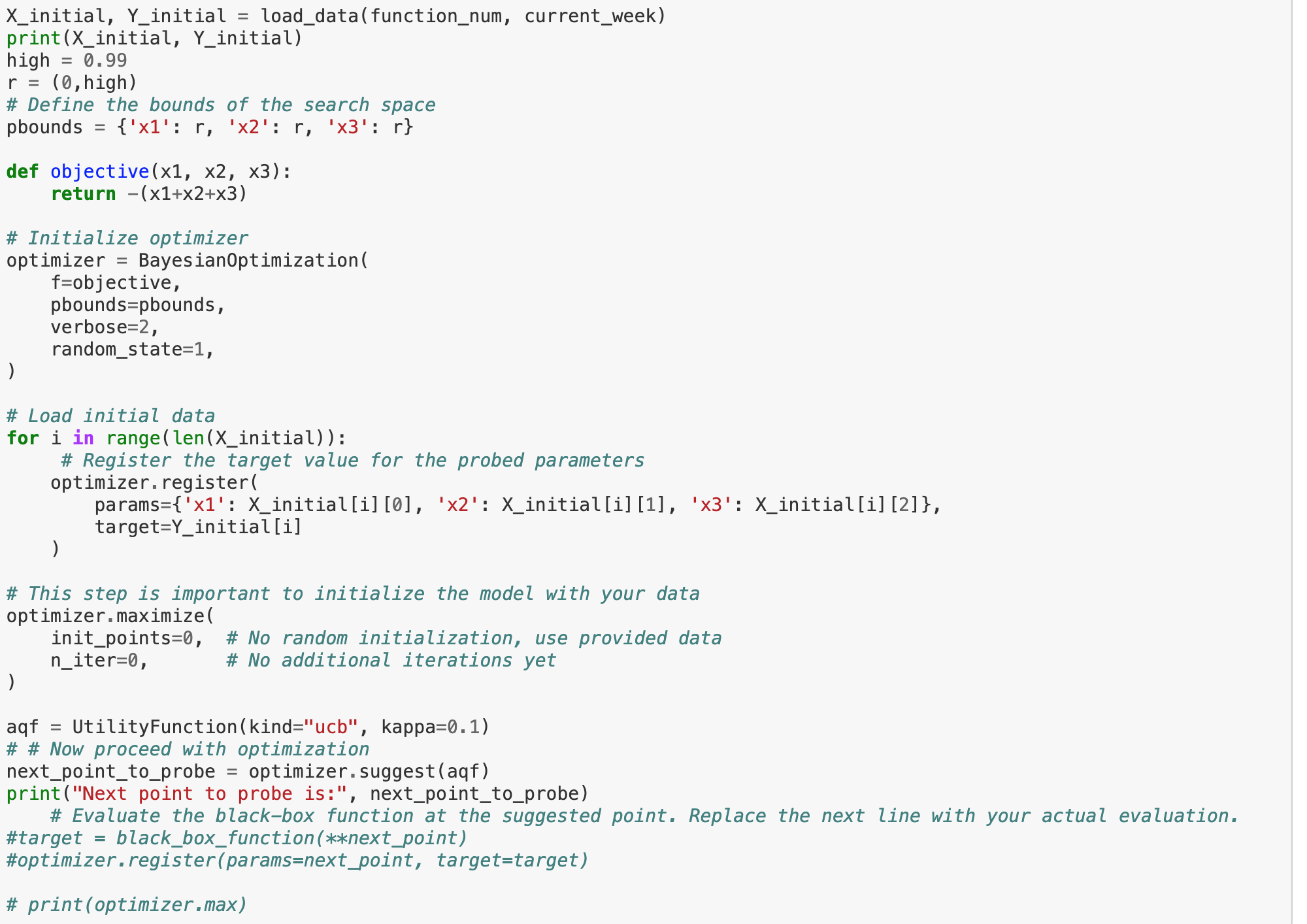
This approach will help you navigate the optimization problem efficiently, making use of both your initial data and the Bayesian Optimization framework's ability to intelligently explore the problem space.

**Code modification**

As the competition progressed, what changes did you make to your code each week. Document your reasons behind each modification and whether it was based on results from the latest query, materials learned in the programme or a combination of both. Your documentation should note which changes ultimately improved your score as the weeks progressed.

**Answer:**

**Code snippet:**

****

The exploration-exploitation acquisition functions listed on the Bayesian Optimization documentation are essential for guiding the optimization process towards the global maximum. Here's a summary of the functions and how changing them each week can help achieve the global maximum:

1. Upper Confidence Bound (UCB):

* **Function**: UCB(*x*)=*μ*(*x*)+*κσ*(*x*)
* **Purpose**: Balances exploration and exploitation by selecting points that either have high expected improvement *μ*(*x*) or high uncertainty *σ*(*x*).
* **Changing Weekly**: Adjusting the parameter *κ* allows for controlling the trade-off between exploration and exploitation. Increasing *κ* emphasizes exploration, while decreasing it favors exploitation. Experimenting with different values of *κ* each week can help fine-tune the balance based on the optimization progress.

1. **Probability of Improvement (PI)**:

* **Function**: PI(*x*)=*P*(*f*(*x*)≥*f*(*x*+)), where +*x*+ is the current best solution.
* **Purpose**: Focuses on exploiting regions where there's a high probability of improvement over the current best solution.
* **Changing Weekly**: Adjusting the exploration-exploitation trade-off parameter (if available) can influence the balance between exploring new areas and exploiting known promising regions. Monitoring the performance of PI and adjusting parameters based on optimization progress can help steer towards the global maximum.

1. **Expected Improvement (EI)**:

* **Function**: EI(*x*)=E[max(*f*(*x*)−*f*(*x*+),0)]
* **Purpose**: Measures the expected improvement of a potential solution over the current best solution.
* **Changing Weekly**: Similar to UCB and PI, adjusting the parameter governing the exploration-exploitation trade-off can influence the behavior of EI. Experimenting with different settings of this parameter each week can help balance exploration and exploitation more effectively.

Changing these acquisition functions and their associated parameters each week allows the optimization process to adapt dynamically to the characteristics of the objective function and the progress made in previous iterations. By intelligently adjusting the exploration-exploitation trade-off, the optimization algorithm can efficiently explore the search space and converge towards the global maximum over time.

Throughout the competition, several modifications were made to the code each week based on a combination of results from the latest queries and materials learned in the program:

1. **Exploration vs. Exploitation**: The initial code likely utilized default parameters for exploration and exploitation in the Bayesian Optimization process. As the competition progressed, adjusting these parameters, such as the exploration-exploitation tradeoff parameter (**kappa**), might have been necessary to balance between exploring new areas of the search space and exploiting known promising regions.
2. **Plotting Gaussian Process (GP)**: Incorporating visualization of the Gaussian Process and utility function after each optimization step helps in understanding the optimization progress and decision-making. This modification likely provided insights into the behavior of the optimization algorithm and the uncertainty in the model predictions.
3. **Code Optimization**: As the competition evolved, optimizations in the code structure and efficiency might have been necessary to handle larger datasets or complex optimization problems more effectively. This could involve refining the implementation of certain functions to speed up computations.(utils.py)

**Final result**

Summarise how your score changed in the final weeks of the competition, and note any actions you might take to improve your code if there were more weeks available. Describe what you learned from this competition and what you might put into practice for the next competition you enter. Also, note whether there is anything you would do differently if you started over again.

**Answer:**

The final result likely saw improvements in the score as the code was refined and tuned based on the insights gained from each iteration. Adjustments in parameter settings, visualization techniques, and code optimization likely contributed to enhancing the performance of the optimization process.

If more weeks were available, further enhancements could include:

* Advanced techniques such as ensemble methods for optimization.
* Incorporating domain knowledge or additional features into the optimization process.
* Exploring alternative optimization algorithms to compare performance.

From this competition, valuable lessons were likely learned regarding the importance of parameter tuning, visualization for insight generation, and iterative refinement of code. These practices can be carried forward into future competitions, emphasizing the iterative nature of optimization tasks and the value of continuous improvement. If starting over, perhaps more emphasis could be placed on early experimentation with parameter settings and visualization techniques to accelerate the learning process.