**Use Case # 1:**

**Optimize material performances** – The design of materials consists of minimizing, maximizing or targeting specific properties. For example, a goal in alloys design is to determine a set of compositions, temperature and pressure that maximizes functions which depend on phases, volume fractions and on the thermodynamic, structural, dynamics, thermal transport and surface properties.

|  |  |
| --- | --- |
| **User:** | Material Scientist/Engineer/Researcher who has knowledge of the design space/parameters and may have a basic knowledge of Python |
| **Function:** | User feeds the initial data (temperature, pressure, composition, etc.) and parameter bounds into the selected optimizer and runs the program (repeats the process iteratively with newly obtained data) |
| **Results:** | Optimizer is executed iteratively by comparing previously obtained solutions until an optimum solution is found by reducing the distribution over possible functions (posterior). |
| **Component:** | Bayes\_Opt - Bayesian Optimization can be used to guide the choice of experiments during materials design and discovery to find good material designs in as few experiments as possible. Bayesian optimization incorporates prior belief about the unknown function and updates the prior with samples drawn from this unknown function to get a posterior that better approximates the unknown function. The model used for approximating the objective function is called *surrogate model* (Gaussian processes, or GP, is used as the surrogate model for Bayes\_Opt). Bayesian optimization also uses an *acquisition function* that directs sampling to areas where an improvement over the current best observation is likely (for Bayes\_Opt, Upper Confidence Bound, or UCB is used). |

**Use Case # 2:**

|  |  |
| --- | --- |
| **User:** | Material Scientist/Engineer/Researcher who wants to optimize an experiment and prioritizes speed vs accuracy. |
| **Function:** | User selects an optimizer that prioritizes speed vs accuracy, and the algorithm will select the correct optimizer that fits the user’s needs |
| **Results:** | The algorithm will be trained with a few optimizers, so it will pick the one that fits the user’s needs. In this case it will most likely be the random optimizer. |
| **Component:** | The random optimizer samples datapoints and ranks them according to the highest output. It is fast but not accurate.  The algorithm (KNN, SVM, or multilinear regression) that chooses the best optimizer has been trained with data from several optimizers. It will determine which one fits the users needs. |

**Use Case # 3:**

**Optimizing oil interfacial tension(IFT) –** Minimizing the oil surface tension enhances the recovery of oil from the bottom of the earth. Oil surface tension depends on biosurfactant concentration, PH, Temperature and the fluid property. Therefore, the main purpose is to have the optimum oil interfacial tension with lowest cost possible (optimum biosurfactant concentration).

|  |  |
| --- | --- |
| **User:** | Scientist/ Engineer/Researcher who has interest on oil related application and may have a basic knowledge of Python |
| **Function:** | User feeds the initial data (biosurfactant concentration, Temperature, composition, etc.) and parameter bounds into the selected optimizer and runs the program. |
| **Results:** | Once a problem is described in mathematical terms, PuLP can be used to find the optimal values. In IFT, the optimization objective is to satisfy the minimum interfacial tension based on least cost. |
| **Component:** | The inequality relations are all linear in nature i.e. the variables are multiplied by constant coefficients and the resulting terms are bounded by constant limits and that’s what makes this problem solvable by PuLP technique. PuLP programming is a technique for the [optimization](https://en.wikipedia.org/wiki/Mathematical_optimization) of a [linear](https://en.wikipedia.org/wiki/Linear) [objective function](https://en.wikipedia.org/wiki/Objective_function), subject to [linear equality](https://en.wikipedia.org/wiki/Linear_equality) and [linear inequality](https://en.wikipedia.org/wiki/Linear_inequality) [constraints](https://en.wikipedia.org/wiki/Constraint_(mathematics)). Its [feasible region](https://en.wikipedia.org/wiki/Feasible_region) is a [convex polytope](https://en.wikipedia.org/wiki/Convex_polytope), which is a set defined as the [intersection](https://en.wikipedia.org/wiki/Intersection_(mathematics)) of finitely many [half spaces](https://en.wikipedia.org/wiki/Half-space_(geometry)), each of which is defined by a linear inequality. Its objective function is a [real](https://en.wikipedia.org/wiki/Real_number)-valued [affine (linear) function](https://en.wikipedia.org/wiki/Affine_function) defined on this polyhedron. A linear programming [algorithm](https://en.wikipedia.org/wiki/Algorithm) finds a point in the [polytope](https://en.wikipedia.org/wiki/Polytope) where this function has the smallest (or largest) value if such a point exists. |

**Use Case # 4:**

**Use Case # 5:**