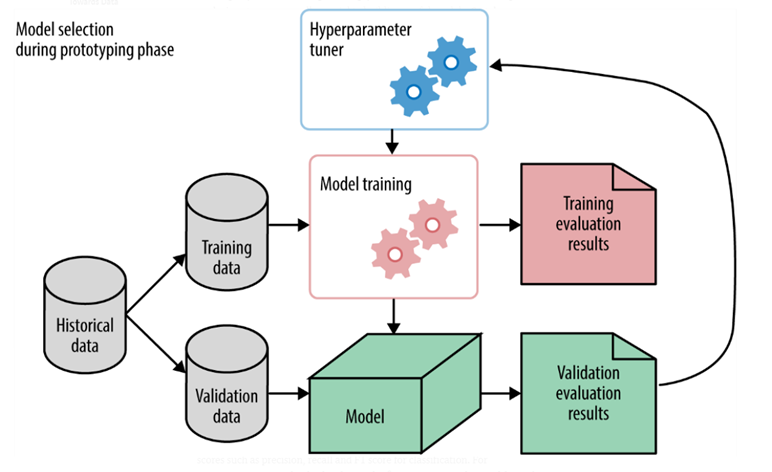
**Hyperparameter Optimizer**

Now, let’s see where will a hyperparameter optimizer stands in the whole process of learning the algorithm.



As can be seen in the above figure [1], the hyperparameter tuner is external to the model and the tuning is done before model training. The result of the tuning process is the optimal values of hyperparameters which is then fed to the model training stage. Let me now introduce Optuna, an optimization library in Python that can be employed for hyperparameter optimization.

**Optuna**

Optuna is a software framework for automating the optimization process of these hyperparameters. It automatically finds optimal hyperparameter values by making use of different samplers such as grid search, random, bayesian, and evolutionary algorithms. Let me first briefly describe the different samplers available in optuna.

* Grid Search: The search space of each hyper-parameter is discretized. The optimizer launches learning for each of the hyper-parameter configurations and selects the best at the end.
* Random: Randomly samples the search space and continues until the stopping criteria are met.
* Bayesian: Probabilistic model-based approach for finding the optimal hyperparameters
* Evolutionary algorithms: •Meta-heuristic approaches that employ the value of the fitness function to find the optimal hyperparameters.

Why Optuna? The following features of optuna encouraged me to use it for hyperparameter tuning for the problems I was trying to solve!

* Eager dynamic search spaces
* Efficient sampling and pruning algorithms
* Easy integration
* Good visualizations
* Distributed optimization

Let us now discuss an example.

**Hyperparameter Optimization of Random Forest using Optuna**

Nw, let’s see how to do optimization with optuna. I’m using the iris dataset to demonstrate this. First, we have to decide the metric based on which we have to optimize the hyperparameters. This metric is thus the optimization objective. Here, I have taken the metric as a cross-validation score with k=3. So let’s first create the objective function.

So, in this case, I’m using the Random forest to make the classification. I wish to optimize only two  – n\_estimators and max\_depth. So, I defined the search space for these hyperparameters. As seen, n\_estimators are integer ranging from 2 to 20, and max\_depth is taken from log uniform ranging from 1 to 32. These are the values specified by me.

You can change these values as well as the types depending on the parameter nature. So this function takes a trial object as its argument. The trial is just one iteration of the optimization experiment. So here my objective is to maximize the cross-validation score and determine what values of n\_estimators and max\_depth would maximize my cross\_val\_score!

The objective function value is evaluated using a study object.

Here direction can be ‘maximize’ or ‘minimize’ depending upon the nature of the objective. I want to maximize the cross\_val\_score, hence I gave ‘maximize’. Also, I have given the no of trials as 100. I haven’t specified the sampler to be used here, by default it is the bayesian optimizer. Once we call the optimize method, it can be seen that the optimization process starts.

Once all trials are completed, from the study object we can find the best trial and also the optimal values of the hyperparameters.

 So, it can be seen that the best accuracy is 0.98 and the best hyperparameter values for n\_estimators and max\_depth are 4 and 2.718 respectively.

Now, let’s explore the different visualization plots available in optuna. Mainly 4 types of plots are relevant from a hyperparameter optimization point of view – history plot, slice plot, contour plot, and parallel coordinate plot. The plots are self-explanatory.

Optuna is a very vast optimization library with applications not just limited to hyperparameter tuning. Through this blog, I just tried to portray a simple example of hyperparameter optimization with Optun