Chapter 9
Learning to Analyze the Pareto-Optimal Front

As mentioned in the previous chapters, evolutionary multi- and many-objective optimization algorithms (EMâOAs) attempt to find a set of well-converged and well-diversified solutions to approximate the true Pareto front (PF). In general, a uniform distribution of solutions across PF is desired. However, this cannot be guaranteed due to the stochasticity involved in EMâOAs. In a contrasting scenario, even a biased distribution of solutions across PF, with a higher concentration of solutions in specific parts of PF, may be desired by the decision maker for a subsequent multi-criterion decision-making (MCDM) task. indexMulti-criterion decision-making (MCDM) To meet such requirements, this chapter presents a machine learning (ML)-based approach, which treats a given PF-approximation as input and trains an ML model to capture the relationship between pseudo-weight vectors derived from the objective vectors in the PF-approximation (F in $\mathcal{Z}$), and their underlying variable vectors (X in $\mathcal{X}$). Subsequently, the trained ML model is applied to predict the solution's X vector for any desired pseudo-weight vector. In other words, the trained ML model is used to create new non-dominated solutions in any desired region of the obtained PF-approximation. Such new solutions could be created to fill apparent gaps in the input PF-approximation toward a more uniform distribution or to enhance the concentration of solutions as desired by the decision maker. The working and usefulness of the above post-optimality analysis basis approach have been demonstrated over several problem instances. However, this approach also has the potential to be integrated within an EMâOA to arrive at the desired distribution.

In principle, the proposed approach, summarized above, conforms with any conventional ML approach in which input–output relationships are learned from the training-dataset, and the resulting ML model is used to predict the output from an unseen given input. In the current context, the training-dataset is based on non-dominated solutions obtained by the EMâOA, in which the input is a unique indicator of a solution (in the objective space or $\mathcal{Z}$ space) and the output is the X vector of the solution. In that scenario, the pseudo-weight vectors [2], denoted by W, have been used as the unique indicator. An advantage of using a pseudo-weight vector is that

D. K. Saxena et al., *Machine Learning Assisted Evolutionary Multi- and Many-Objective Optimization*, Genetic and Evolutionary Computation,
https://doi.org/10.1007/978-981-99-2096-9_9

each component lies in the [0, 1] range and therefore does not require normalization. Once the ML model is trained, any new pseudo-weight vector can be used as input, to find the corresponding solution's X vector. Subsequently, its F vector can be computed using the objective formulations stated in the given multi- or many-objective optimization problem (MOP/MaOP).

The remainder of this chapter is organized as follows. The ML-based procedure for the post-optimal *PF* analysis is described in Sect. 9.1. The results on multi- and many-objective test problems (constrained and unconstrained) and real-world problems are presented in Sect. 9.2. Finally, Sect. 9.4 summarizes the findings.

9.1 ML-Based *PF* Analysis: Proposition and Validation

As discussed in previous chapters, using an EMâOA with N population members to solve an MOP/MaOP can lead to N potentially non-dominated solutions $X^{(k)} \in \mathbb{R}^n$, for $k = 1, 2, \ldots, N$, along with their respective objective vectors $F(X^{(k)}) \in \mathbb{R}^M$, for $k = 1, 2, \ldots, N$. In the $\mathcal{Z}$ space, these objective vectors are referred to as the *PF*-approximation. From the position of each objective vector on the *PF*-approximation, the corresponding pseudo-weight vector $W^{(k)} = \left(w_1^{(k)}, w_2^{(k)}, \ldots, w_M^{(k)}\right)$; $W^{(k)} \in \mathbb{R}^M$ can be computed as follows [2]:

$$w_i^{(k)} = \frac{\left(f_i^{\max} - f_i(X^{(k)})\right) \Big/ \left(f_i^{\max} - f_i^{\min}\right)}{\sum_{j=1}^{M} \left(f_j^{\max} - f_j(X^{(k)})\right) \Big/ \left(f_j^{\max} - f_j^{\min}\right)}, \tag{9.1}$$

where $f_i^{\max}$ and $f_i^{\min}$ are the maximum and minimum values in the ith objective, among all the non-dominated solutions in the *PF*-approximation. The pseudo-weight (W) vector for any solution can be viewed as a representative identity on the obtained *PF*-approximation. For example, given a two-objective problem, the best solution in f_1 would correspond to a vector W of (1, 0), which means that the solution is of 100% importance with respect to f_1, and of no importance with respect to f_2.

Clearly, Eq. 9.1 indicates that the W vectors are linearly derived from the F vectors. If the solutions are sorted according to their F vectors, the respective W vectors also are sorted accordingly. Previous studies on the concept of *innovization* (Chap. 3) have revealed that a set of Pareto-optimal solutions usually possesses certain patterns or constancy with respect to certain variables. Since a W vector is derived from an F vector that bears an exact relationship with its underlying X vector, it is fair to assume a linkage between a W vector and the corresponding X vector. In this context, this study captures the relationships between the derived W vectors and corresponding X vectors from a *PF*-approximation, using an ML method. Once these patterns, if any, are captured, the desired $\bar{W}$ vectors can be used to obtain the $\bar{X}$ vectors of new (potentially non-dominated) solutions. Figure 9.1 illustrates the

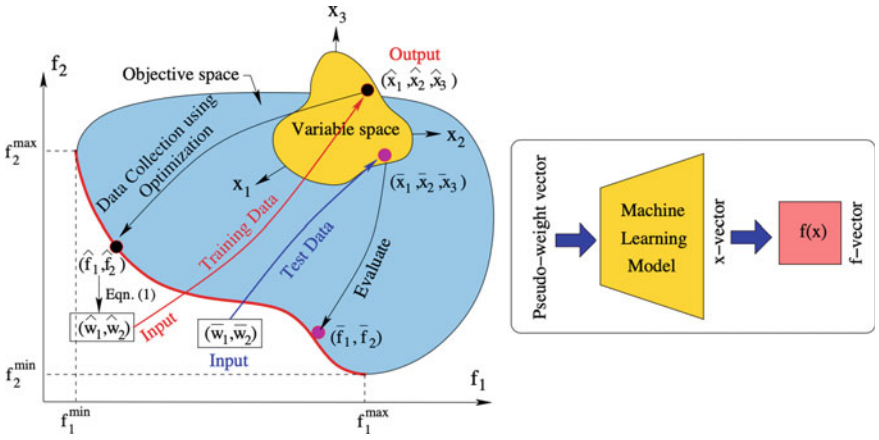


Fig. 9.1 Training data generation and test-data to create a new potentially Pareto-optimal solution (taken from [3])

underlying ML training and testing procedure, as required, which is detailed in the following steps:

Step 1: From each solution in the obtained *PF*-approximation, compute the W vectors using Eq. 9.1.

Step 2: Construct the training-dataset $[W^{(k)}, X^{(k)}]\ \forall\ k = 1, 2, \ldots, N$, with $W^{(k)}$ as input and $X^{(k)}$ as the corresponding target.

Step 3: Train an ML model using the training-dataset.

Step 4: Use the trained ML model to find $\bar{X}$ for any specific desired $\bar{W}$ vector. Then, compute $\bar{F}$ from $\bar{X}$ using the original objectives' formulation, where $\bar{F} = F(\bar{X})$.

The resulting trained ML model can then be used for several different tasks, as summarized below.

Task 1: Gap-filling: The trained ML model can be used to find new and potentially non-dominated solutions, so that the apparent gaps in the obtained *PF*-approximation can be filled. This could be realized by creating new W vectors in those gaps and then applying the trained ML model to create new (non-dominated) solutions.

Task 2: Gap-validation: The trained ML model can be used to determine whether or not an apparent gap in the *PF*-approximation is a natural gap in the true *PF*. This could be realized by first creating new W vectors in the apparent gaps; obtaining the corresponding X vectors through the trained ML model; evaluating these X vectors to obtain the corresponding F vectors; and investigating whether or not these F vectors are dominated by any solution in the existing *PF*-approximation. If any such F vector

stands dominated, it implies that the apparent gap for which the W vector was created happens to be a natural gap in the true PF.

Task 3: Preferred solution-density propagation: The trained ML model can be used to populate new non-dominated solutions in any preferred region of the PF-approximation, toward ease of visualization and decision-making. This could be achieved by creating suitable W vectors in the region of interest and then using the trained ML model to find new and potentially non-dominated solutions in that region.

Task 4: Efficient offspring creation: The trained ML model can be used within an EMâOA run, to create new offspring solutions. This could be realized by first training an ML model using the non-dominated solutions from the parent population; selecting the W vectors in the less dense areas; and applying the trained ML model to create new offspring solutions (one solution per W vector).

The scope of this proof-of-concept study has been restricted to the first three tasks mentioned above since the last task requires the development of a new EMâOA.

9.1.1 Chosen ML Methods

In this section, details of the ML methods utilized in this study are discussed. The problem statement at hand is the task of predicting an n-dimensional X vector from a given M-dimensional W vector. Since, in general, $M \ll n$, developing an accurate ML model with few input parameters and a large number of output parameters is a challenging task. Two different ML methods have been used for this task, namely deep neural network (DNN) [6] and Gaussian process regression (GPR) [13]. In both the DNN-based and the GPR-based approaches, the X vectors of the solutions are normalized to zero mean and unit variance a priori, as required by the ML methods. As mentioned at the beginning of this chapter, the W vectors are already within the [0, 1] range and therefore do not require normalization.

For each problem considered here, PF is approximated using NSGA-III, with the population size given by $N = (110M + 10)$. These population sizes were selected based on a trial-and-error study. From these N solutions, randomly selected $100M$ solutions are used as the training-dataset, and the other $10M$ solutions are used as the test-dataset. Subsequently, the remaining 10 solutions are used as the validation-dataset as required by the DNN, but are discarded in the case of GPR, since GPR does not require a validation step. The validation-dataset is uniformly sampled from the PF-approximation for appropriate model selection in the case of DNN. Notably, the same population size has been used for both to maintain the similarity of the training-dataset size. Furthermore, this approach is not conditional on the source of the training-dataset. Therefore, the training-dataset obtained from the non-dominated solutions of the PF-approximation can be replaced by the non-dominated solutions at the end of any intermediate generation of an EMâOA run.

9.1.1.1 DNN-Based Approach

For the DNN-based approach, multi-layer perceptrons have been implemented using the PyTorch library [12], with W vectors as inputs and their underlying X vectors as respective targets, and the corresponding hyper-parameters are tuned using Optuna [1]. Due to the proof-of-concept nature of the study, DNNs with hidden layers ranging from 1 to 6 are used with 'ReLU' as activation function, and 'Adam' [9] as optimizer. The complexity of the DNN architecture and granularity of hyper-parameters can be further increased for handling more complicated problems.

9.1.1.2 GPR-Based Approach

An approach similar to surrogate modeling is used to train the GPR model. In that approach, each component of the target—that is, $x_i \in X$—is modeled independently. A diverse set of kernels, mean functions, and other hyper-parameters is considered in a grid-search for finding the most suitable setting.

9.1.2 Handling Variable Bounds and Constraints

In any MOP/MaOP, the variables are usually restricted within the pre-specified lower and upper bounds. Since a DNN or GPR model usually does not restrict its output values (X vector) within any bounds by default, the resulting output values for any test-dataset can be outside of those bounds, if specific steps are not taken. Hence, the target values in the training-dataset have also been normalized. Subsequently, the output X vector from the trained ML model is denormalized and clipped to within the specified lower and upper bounds.

Furthermore, constraint satisfaction is also a strict requirement in solving a constrained MOP/MaOP. In such a scenario, the resulting X vectors from the trained ML model may not guarantee the constraint satisfaction by default. Hence, the infeasible solutions (if created by the ML model) are simply discarded; but a more sophisticated constraint handling method can be used during the ML training. For instance, the constraint value of each constraint can be included as an additional output in the training-dataset. During testing, if any W vector (input) produces a positive constraint value (meaning a constraint violation), the solution can simply be ignored. To implement such constraint handling, the training-dataset must contain some infeasible solutions that are non-dominated to the feasible non-dominated solution set, to effect a better learning. In this study, such a tailored constraint handling approach has not been included.

9.1.3 Test Suite

For validation of ML models and analysis of different tasks (mentioned earlier in Sect. 9.1), several test and real-world problems have been used, including the following:

- Two-objective unconstrained ZDT [16], and constrained OSY [11] and BNH [15] problems.
- Three-objective unconstrained DTLZ [4] and WFG [7] problems.
- Many-objective unconstrained DTLZ [4] and constrained C-DTLZ [8] problems.
- Real-world Carside [5] and Crashworthiness [10] problems.

9.2 Validation of ML Models

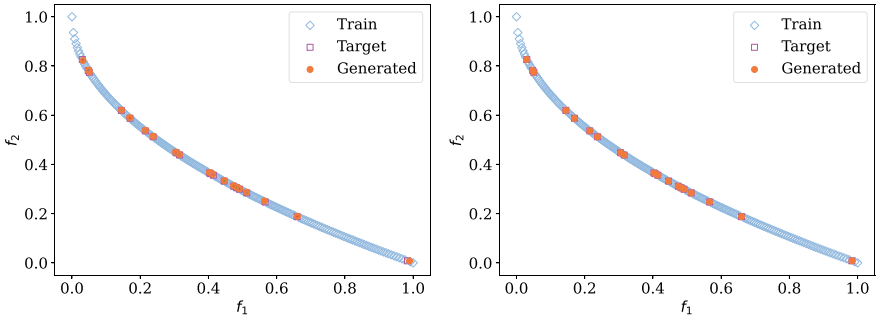
This section focuses on the validation of the trained ML model. Recall that given a PF-approximation with $N = 110M + 10$ solutions, the training-data was constituted by randomly chosen $100M$ solutions. Of the remaining ones, the randomly picked $10M$ solutions constitute the test-data and are referred to as *Random* test-data. Once the W vectors for these test solutions are subjected to the trained ML model (DNN or GPR-based), it yields the X vectors (referred to as model-based X vectors) which on evaluation offer the corresponding model-based F vectors. Since the original X and F vectors of these test solutions are available from the PF-approximation, the mean absolute error (MAE) between the original and model-based X vectors (normalized using the variable bounds), and the original and model-based F vectors (normalized using ideal and nadir points) can be computed and referred to as MAE_X and MAE_F, respectively.

For validation of the ML model, a range of problems is considered, and 11 independently seeded runs of NSGA-III are performed for each problem. The MAE_X resulting from each of the 11 runs can be processed, leading to their respective mean and standard deviation, namely $\mu(MAE_X)$ and $\sigma(MAE_X)$, respectively. Similarly, using the MAE_F values over the 11 runs, the corresponding $\mu(MAE_F)$ and $\sigma(MAE_F)$ can be obtained. Table 9.1 presents the above indicators for a range of two-objective problems, for both DNN- and GPR-based approaches. Evidently:

- Low values of $\mu(MAE_X)$ and $\sigma(MAE_X)$ indicate that the actual and the ML model-based approximated X vectors are reasonably close to each other. The same applies to the actual and approximated F vectors. These observations validate the underlying ML models. A visual evidence for this claim is presented for a sample problem, namely ZDT1, in Fig. 9.2.
- Compared to the DNN-based approach, the GPR-based approach offers lower values for each of $\mu(MAE_X)$ and $\sigma(MAE_X)$, also $\mu(MAE_F)$ and $\sigma(MAE_F)$. Hence, the GPR-based approach is seemingly better than the DNN-based approach.

Table 9.1 Comparison of DNN and GPR, with *Random* test-data on two-objective problems

Problem	M	Model	$\mu(MAE_X)$	$\mu(MAE_F)$	$\sigma(MAE_X)$	$\sigma(MAE_F)$
ZDT1	2	DNN	4.606E-04	6.918E-03	2.575E-04	3.689E-03
		GPR	5.525E-09	3.462E-06	3.064E-09	7.511E-06
ZDT2	2	DNN	7.260E-04	1.387E-02	4.065E-04	6.786E-03
		GPR	1.398E-05	4.200E-04	1.946E-05	6.093E-04
ZDT3	2	DNN	4.619E-04	1.757E-02	2.591E-04	1.434E-02
		GPR	3.659E-06	3.098E-04	4.985E-06	8.355E-04
OSY	2	DNN	7.833E-03	3.607E-02	3.323E-02	3.521E+00
		GPR	1.491E-03	1.273E-03	1.146E-02	3.407E-01



(a) DNN-based approach (b) GPR-based approach

Fig. 9.2 DNN- and GPR-based approaches on *Random* test-dataset for the ZDT1 problem (taken from [3])

The scope of ML-model validation is expanded by including some two-objective (constrained), three-objective, and many-objective problems. However, for experiments, only the GPR-based approach which performed better on the unconstrained two-objective problems has been considered. The results are presented in Table 9.2 along side some self-explanatory representative plots in Figs. 9.3 and 9.4.

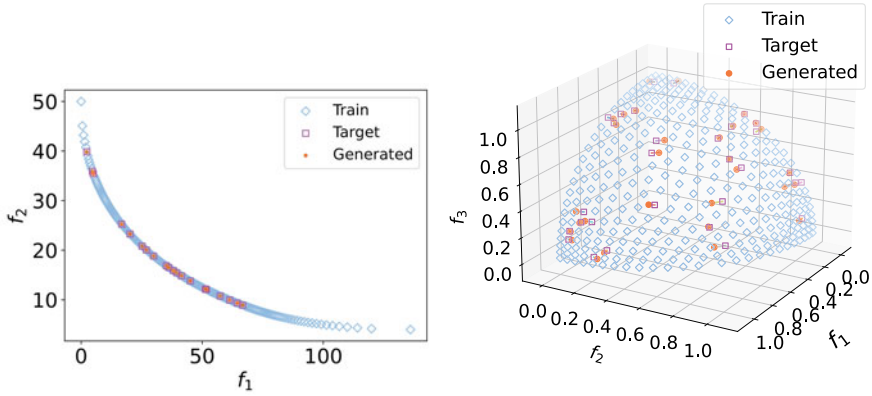
9.3 ML-Based *PF* Analysis for Different Tasks

This section presents the application of the ML-based *PF* analysis toward the first three tasks highlighted in Sect. 9.1. As a pre-requisite, the use of the different test-data types as defined below must be noted.

- *Edge* test-data: It implies that the test solutions representing the gap belong to one of the extreme regions of the *PF*-approximation.
- *Continuous* test-data: It implies that the test solutions representing the gap are bounded by continuous regions in the *PF*-approximation.

Table 9.2 Performance of GPR-based approach on two-objective (constrained), three-objective, and many-objective problems, for *Random* test-dataset

Problem	M	$\mu(MAE_X)$	$\mu(MAE_F)$	$\sigma(MAE_X)$	$\mu(MAE_F)$
BNH	2	1.890E-03	2.073E-04	3.004E-03	9.670E-03
DTLZ2	3	3.419E-03	1.153E-02	2.494E-03	7.168E-03
WFG2	3	4.773E-02	4.922E-02	1.220E-01	1.732E-01
Carside	3	1.171E-02	3.957E-03	9.080E-03	1.471E-02
DTLZ2	5	9.887E-03	1.382E-02	9.155E-03	8.668E-03
	10	4.582E-02	1.418E-02	3.716E-02	8.500E-03
C2-DTLZ2	5	1.020E-02	1.323E-02	1.028E-02	7.448E-03
	10	4.532E-02	1.418E-02	3.552E-02	9.490E-03



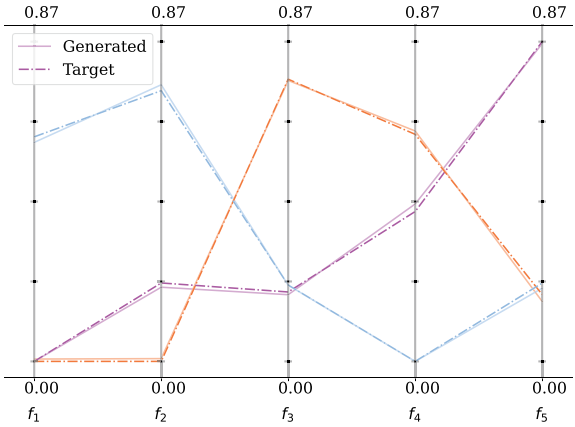
(a) BNH (b) DTLZ2

Fig. 9.3 GPR-based approach on *Random* test-dataset constrained two-objective (BNH) and three-objective (DTLZ2) problems (taken from [3])

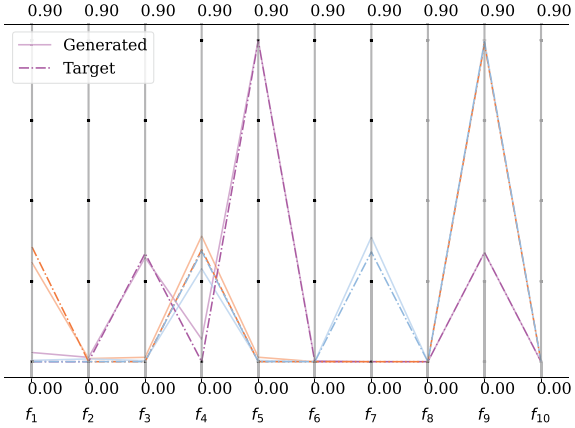
9.3.1 Task 1: Gap-Filling

This section focuses on the validation of Task 1, i.e., the gap-filling task. Unlike in Sect. 9.2 where *Random* test-data was used, here *Edge* test-data has first been used, followed by the *Continuous* test-data. The rest of the procedure is the same as used for validation of the ML model in Sect. 9.2.

Table 9.3 shows the MAE_X and MAE_F indicators on some two-objective problems with *Edge* test-data, for both DNN- and GPR-based approaches. In that, compared to the DNN-based approach, the GPR-based approach offers lower values for each of $\mu(MAE_X)$ and $\sigma(MAE_X)$, and also $\mu(MAE_F)$ and $\sigma(MAE_F)$. This implies that the GPR-based approach is better for finding edge gap solutions. For a visual representation, Fig. 9.5 shows the edge gap solutions produced using DNN- and GPR-based approaches on the ZDT1 problem. As can be observed, while both approaches seem



(a) 5-objective DTLZ2

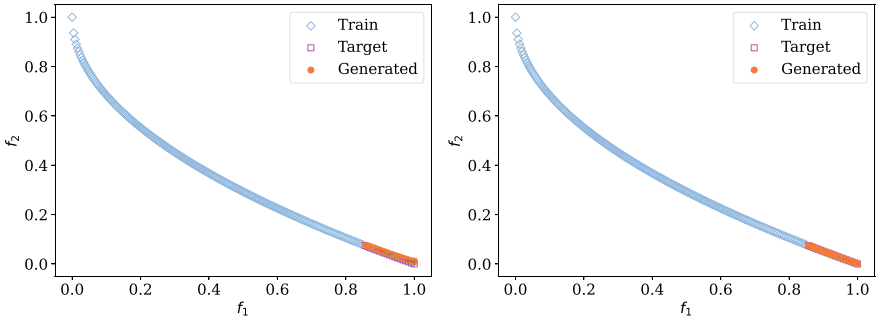


(b) 10-objective C2-DTLZ2

Fig. 9.4 Parallel Coordinate Plots showing that the GPR-based F vectors conform with the F vectors in the PF-approximation (*Randomly* picked), for five- and 10-objective DTLZ2 problems (taken from [3])

Table 9.3 Comparison of DNN and GPR, with *Edge* test-data on two-objective problems

Problem	M	Model	$\mu(MAE_X)$	$\mu(MAE_F)$	$\sigma(MAE_X)$	$\sigma(MAE_F)$
ZDT1	2	DNN	2.012E-03	1.864E-02	4.350E-04	4.332E-03
		GPR	1.086E-06	2.447E-05	8.081E-07	1.820E-05
ZDT2	2	DNN	1.313E-03	3.986E-02	1.207E-04	3.680E-03
		GPR	8.651E-04	3.811E-02	2.308E-04	1.023E-02



(a) DNN-based approach (b) GPR-based approach

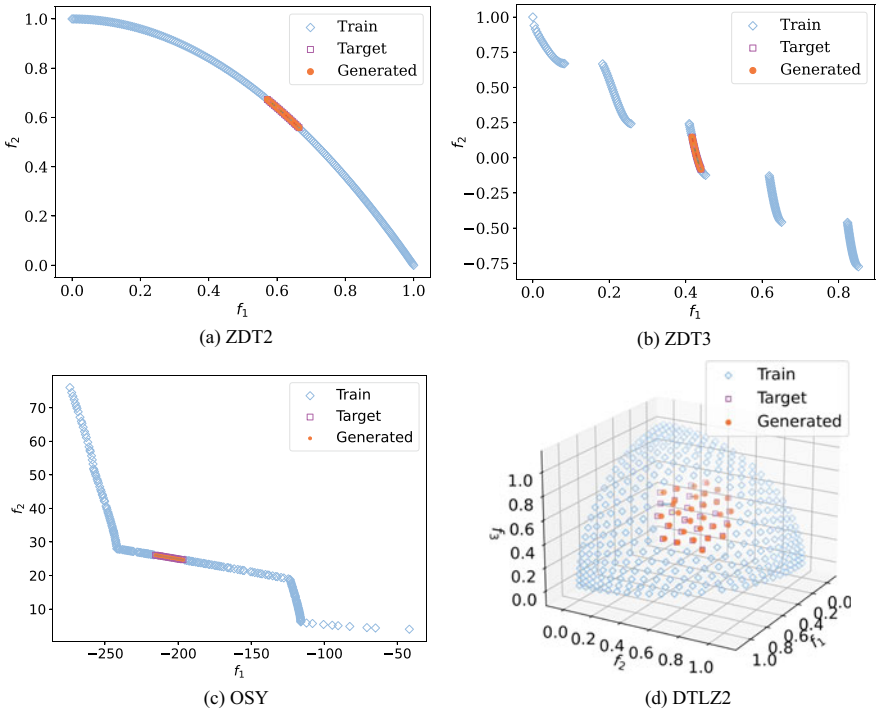
Fig. 9.5 DNN- and GPR-based approaches on *Edge* test-dataset for the ZDT1 problem (taken from [3])

Table 9.4 Comparison of DNN and GPR, with *Continuous* test-data on two-objective problems

Problem	M	Model	$\mu(MAE_X)$	$\mu(MAE_F)$	$\sigma(MAE_X)$	$\sigma(MAE_F)$
ZDT1	2	DNN	6.281E-04	9.264E-03	1.051E-04	1.374E-03
		GPR	2.625E-08	8.466E-08	5.694E-08	1.806E-07
ZDT2	2	DNN	7.387E-04	1.321E-02	1.242E-04	2.357E-03
		GPR	1.083E-04	3.633E-03	3.903E-05	1.320E-03
ZDT3	2	DNN	1.026E-03	3.948E-02	2.030E-04	1.994E-02
		GPR	3.182E-05	3.365E-03	9.706E-06	2.015E-03
BNH	2	DNN	3.701E-03	2.760E-03	4.185E-03	5.378E-02
		GPR	2.343E-03	1.274E-04	3.054E-03	2.224E-03
OSY	2	DNN	2.172E-03	2.720E-02	3.685E-03	6.518E-01
		GPR	1.471E-03	1.243E-03	4.562E-03	5.371E-02

efficient in finding edge gap solutions, the GPR-based approach produces slightly better (more converged) solutions.

Extending the above analysis to *Continuous* test-data, Table 9.4 shows the MAE_X and MAE_F indicators on some two-objective problems, for both DNN- and GPR-based approaches. As evident, compared to the DNN-based approach, the GPR-based approach offers lower values for each of $\mu(MAE_X)$ and $\sigma(MAE_X)$, and also $\mu(MAE_F)$ and $\sigma(MAE_F)$. This implies that the GPR-based approach is better for finding continuous gap solutions as well. For a visual representation, Fig. 9.6 shows the continuous gap solutions produced using the better performing GPR-based approach.



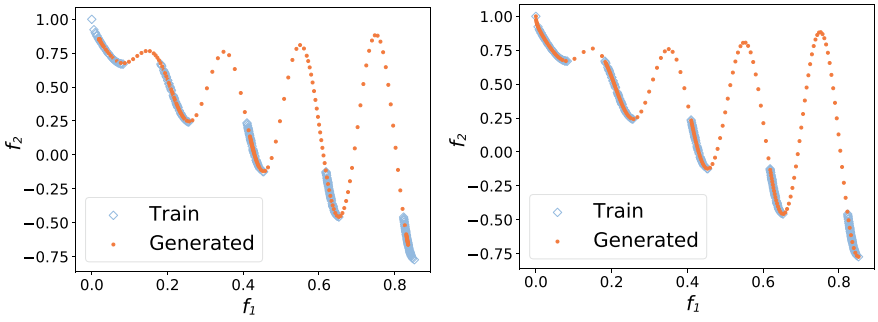
(a) ZDT2 (b) ZDT3

(c) OSY (d) DTLZ2

Fig. 9.6 GPR-based approach on *Continuous* test-dataset two-objective (ZDT2 and ZDT3), constrained two-objective (OSY) and three-objective (DTLZ2) problems (taken from [3])

9.3.2 Task 2: Gap-Validation

This section focuses on the validation of Task 2, i.e., the gap-validation task. For this task, it is imperative to select a problem which has natural gaps in its true *PF*, so that it can be validated. Hence, the ZDT3 problem has been used for validating this task. Toward it, a sample set of 200 uniformly distributed *W* vectors has been created across the obtained *PF*-approximation; the trained ML models have been applied to find the respective *X* vectors; and their corresponding *F* vectors are computed. The resulting *F* vectors are shown in Fig. 9.7 (in red color), along with the originally obtained *PF* approximation (in blue color), for both DNN- and GPR-based approaches. It is evident that the *W* vectors corresponding to the true gaps produce dominated solutions, but those corresponding to parts of the true *PF* result in non-dominated solutions. Therefore, this procedure can also be used to testify whether the apparent gaps in the *PF*-approximation offered by an EMâOA are natural gaps or not.



(a) DNN on ZDT3 (b) GPR on ZDT3

Fig. 9.7 Pseudo-weights on true gaps produce dominated solutions, but pseudo-weights on true *PF* produce non-dominated solutions, demonstrating Task 2 (taken from [3])

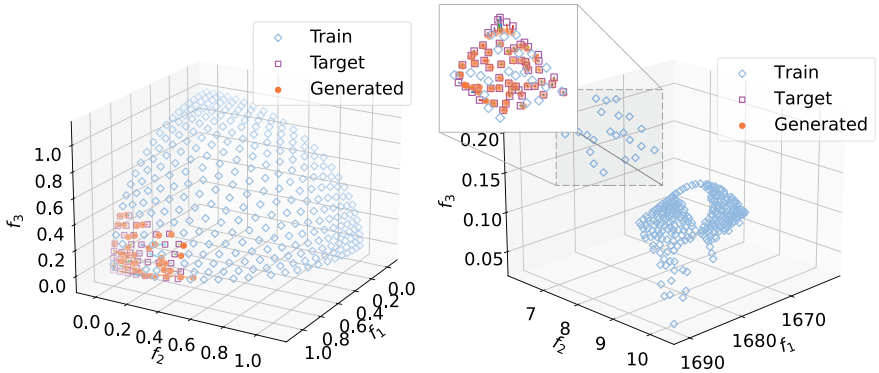
9.3.3 Task 3: Preferred Solution-Density Propagation

This section focuses on validation of Task 3, i.e., preferred solution-density propagation. In other words, the aim of this task is to improve the density of low-density regions of a given *PF*-approximation. In order to simulate a scenario in which an EMâOA produces a non-uniformly distributed *PF*-approximation, some solutions are removed from a specific region of a uniformly distributed *PF*-approximation. In such a scenario, a GPR model can still be trained, and W vectors from the low-density region of the *PF*-approximation can be used to produce new solutions in that region. Figure 9.8 reveals that the GPR-based approach can provide additional solutions in such low-density regions for three-objective test (DTLZ2) and real-world (crashworthiness) problems.

9.4 Summary

In this chapter, it has been demonstrated that ML models can be used to learn patterns between pseudo-weight vectors and corresponding variable vectors and generate new points on the *PF*-approximation without doing any additional optimization tasks. Furthermore, it has also been demonstrated that this approach can scale up well to handle many-objective test problems, and also real-world problems. This proof-of-concept study and a recent extended study [14] have paved the way for encapsulation of this approach as an additional operator in an EMâOA, to achieve a better distributed *PF*-approximation. Owing to the fact that the ML models are conditioned on the pseudo-weight vectors, this approach can be readily used for decision-making by the user without the need for further optimization.

In the future, it would be interesting to compare this approach with optimization-based gap-filling methods, including preference-based EMâOAs. Applications to



(a) DTLZ2 (b) Crashworthiness

Fig. 9.8 Additional solutions are supplied by the GPR-based approach for two three-objective problems in regions of low density of solutions, demonstrating Task 3. A few blue points were found by EMâOA on a part of the *PF*-approximation, but this ML approach has nicely replenished them (taken from [3])

more complex real-world problems will further shed light on the potential of this approach. Furthermore, such modeling approaches can be improved to include learning of constraint violation and variable bounds during the training process itself. For example, specific activation functions (such as ReLU) can be used for the output layer of DNNs to restrict the output within the variable bounds. Although constraint satisfaction has always occurred in the results here, it would be a challenging task to include constraint satisfaction in the training process, since all obtained solutions (training-dataset) are expected to be feasible. Nevertheless, this proof-of-concept study opens up a unique use of ML methods in assisting multi- and many-objective optimization.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19, pp. 2623–2631. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3292500.3330701
2. Deb, K.: Multi-objective Optimization using Evolutionary Algorithms. Wiley, Chichester, UK (2001)
3. Deb, K., Gondkar, A., Anirudh, S.: Learning to predict Pareto-optimal solutions from pseudo-weights. In: Emmerich, M., Deutz, A., Wang, H., Kononova, A.V., Naujoks, B., Li, K., Miettinen, K., Yevseyeva, I. (eds.) Evolutionary Multi-criterion Optimization, pp. 191–204. Springer Nature Switzerland, Cham (2023)
4. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable test problems for evolutionary multiobjective optimization. In: Evolutionary Multiobjective Optimization: Theoretical Advances

and Applications, pp. 105–145. Springer London, London (2005). https://doi.org/10.1007/1-84628-137-7_6
5. Gu, L., Yang, R.J., Tho, C.H., Makowski, L., Faruque, O., Li, Y.: Optimization and robustness for crashworthiness of side impact. Int. J. Veh. Des. **26**(4) (2001)
6. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Comput. **18**(7), 1527–1554 (2006). https://doi.org/10.1162/neco.2006.18.7.1527
7. Huband, S., Hingston, P., Barone, L., While, L.: A review of multiobjective test problems and a scalable test problem toolkit. IEEE Trans. Evol. Comput. **10**(5), 477–506 (2006)
8. Jain, H., Deb, K.: An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, Part II: handling constraints and extending to an adaptive approach. IEEE Trans. Evol. Comput. **18**(4), 602–622 (2014). https://doi.org/10.1109/TEVC.2013.2281534
9. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015, Conference Track Proceedings (2015). arxiv:abs/1412.6980
10. Liao, X., Li, Q., Zhang, W., Yang, X.: Multiobjective optimization for crash safety design of vehicle using stepwise regression model. Struct. Multidiscip. Optim. **35**, 561–569 (2008)
11. Osyczka, A., Kundu, S.: A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm. Struct. Optim. **10**, 94–99 (1995). https://doi.org/10.1007/BF01743536
12. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol. 32, pp. 8024–8035. Curran Associates, Inc. (2019)
13. Rasmussen, C.E.: Gaussian Processes in Machine Learning, pp. 63–71. Springer, Berlin Heidelberg (2004). https://doi.org/10.1007/978-3-540-28650-9_4
14. Suresh, A., Deb, K.: Machine learning based prediction of new Pareto-optimal solutions from pseudo-weights. IEEE Trans. Evol. Comput. (in press)
15. To, T.B., Korn, B.: Mobes: A multiobjective evolution strategy for constrained optimization problems. In: Proceedings of the Third International Conference on Genetic Algorithms (MENDEL97), pp. 176–182 (1997)
16. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: empirical results. Evol. Comput. **8**(2), 173–195 (2000). https://doi.org/10.1162/106365600568202