Machine Learning - Software Project

Component 5

Authors: Liviu-Ștefan Neacșu-Miclea, Răzvan-Gabriel Petec
Specialization: Applied Computational Intelligence
Group: 246/2

1 Model details

1.1 SVR

This work implements the ϵ-Support Vector Regression (ϵ-SVR). The learning algorithm tries to optimize the dual problem

$$\min_{\alpha^-,\alpha^+} \frac{1}{2}\begin{bmatrix}(\alpha^+)^t & (\alpha^-)^t\end{bmatrix}\begin{bmatrix}Q & -Q\\ -Q & Q\end{bmatrix}\begin{bmatrix}\alpha^+\\ \alpha^-\end{bmatrix} + p^t\begin{bmatrix}\alpha^+\\ \alpha^-\end{bmatrix}$$

$$\text{subject to} \quad \alpha^+ - \alpha^- = 0,\ 0 \le \alpha_i^-, \alpha_i^+ \le C, \forall\ i = \overline{1,n},$$

then the learned hypothesis is $h(x) = \sum_{i=1}^{n}(alpha^+ - alpha^-)K(x_i, x) + b$ [CL11].

This is a Quadratic Program (QP). If a decomposable kernel $K(x,y) = \phi(x)^t\phi(y)$ is used, then Q is a positive semidefinite matrix, because $Q_{ij} = K(x_i, x_j)$, therefore there exists the matrix $\Phi = \begin{bmatrix}\phi(x_1) & \cdots & \phi(x_n)\end{bmatrix}^t$ such that $Q = \Phi^t\Phi$. Additionally, it can be proved that $S = \begin{bmatrix}Q & -Q\\ -Q & Q\end{bmatrix}$ is positive semidefinite. Indeed for any vectors α^+, α^- we have $\begin{bmatrix}(\alpha^+)^t & (\alpha^-)^t\end{bmatrix}\begin{bmatrix}Q & -Q\\ -Q & Q\end{bmatrix}\begin{bmatrix}\alpha^+\\ \alpha^-\end{bmatrix} = (\alpha^+)^tQ(\alpha^+) - (\alpha^+)^tQ(\alpha^-) - (\alpha^-)^tQ(\alpha^+) + (\alpha^-)^tQ(\alpha^-) = (\alpha^+ - \alpha^-)^tQ(\alpha^+ - \alpha^-) \ge 0$, which is true because Q is p.s.d., hence S is also p.s.d. Therefore, the unconstrained version of the optimization problem above is convex, and any local minimum is in fact a global minimum.

It is easily observed that $\alpha = \begin{bmatrix}(\alpha^+)^t & (\alpha^-)^t\end{bmatrix}^t = 0$ is a feasible solution.

1.1.1 Implementation details

This work implements two versions of ϵ-SVR optimizers.

The first one is the Sequential Minimal Optimization algorithm as described by libsvm paper [CL11] and recalled in Component 3. The idea is to iteratively optimize a number of subproblems of two variables x_i, x_j chosen from the big system such that the choice to optimize them improves the global solution. Smola et al. noted that the choice heuristic, namely the WSS-1 algorithm, may sometimes fail, and provided alternative guidelines [SS04]. The current implementation chooses the x_i that does not satisfy the KKT conditions and has the greatest impact on the objective function. If there not exists a valid x_j, or the same pair x_i, x_j was chosen in the previous iteration (which will also be selected the next epochs and prevents the algorithm from improving the solution), the second best choice x_i is considered and the process is repeated until a valid pair is found. If the algorithm still fails, then the first value of x_i is paired with a random choice for x_j in order to keep the algorithm running.

The two-variables subproblem has a following form:

$$\min_{\alpha_i,\alpha_j} f_2(\alpha_i,\alpha_j) = q_1\alpha_i^2 + q_2\alpha_j^2 + q_3\alpha_i\alpha_j + q_4\alpha_i + q5\alpha_j$$
$$s.t. u_i\alpha_i + u_j\alpha_j = D, 0 \leq \alpha_i, \alpha_j \leq C,$$

where $u_i, u_j \in -1, 1$

The equality constraint is the implicit form of the line equation, moreover, parallel to either the principal or the secondary diagonal. The inequality constraints define the surface of a square whose corners are $(0,0),(0,C),(C,0),(C,C)$. The feasible solutions lie on the segment that represent the intersection of the line with the square. By substituting $a_i = -\frac{u_j}{u_i}a_j$ into f_2, we obtain a single variable function $g_2(\alpha_j) = r_1\alpha_j^2 + r_2\alpha_j + r_3$. Finally, we reduce the two-variables subproblem to an easier case of minimizing a one-variable quadratic function on a real interval.

The second implementation of ϵ-SVR sees the problem purely from a QP point of view (as others have also remarked [SS04]), and uses a general purpose non-linear optimization algorithm, namely the smoothing Newton method.

1.1.2 Choosing the bias

The libsvm implementation relies on the stationary point KKT conditions (enforcing it if needed) to determine the bias [CL11]. This work implements the more intuitive approach of averaging the errors between the current prediction and the ground truth:

$$b = \frac{1}{n}\sum_{i=1}^{n}(y_i - \alpha_i K(x_i, x)).$$

1.2 Random Forest

The Random Forest Classifier is an ensemble learning method that combines multiple decision trees to improve the model's predictive performance and robustness. The algorithm builds a collection of decision trees during training and aggregates their predictions (through majority voting for classification) to make the final prediction. This implementation provides flexibility in defining the number of estimators, the maximum depth of trees, feature selection methods, and splitting criteria.

The Random Forest Classifier is an ensemble learning technique that constructs a collection of decision trees during the training phase and aggregates their predictions to improve performance and robustness. It relies on bootstrapping, random feature selection, and majority voting to achieve diversity and accuracy. This section details the implementation of the Random Forest Classifier.

1.2.1 Class Structure and Initialization

The Random Forest Classifier is defined as a class with the following key parameters:

- **n_estimators:** Number of decision trees in the forest (default: 10).
- **max_depth:** Maximum depth of each decision tree (default: no limit).
- **max_features:** Number of features to consider for splitting at each node. Supported values:
  - `sqrt`: Square root of the total number of features.
  - `log2`: Logarithm base 2 of the total number of features.
  - Integer value specifying the number of features.
- **criterion:** Splitting criterion for decision trees, e.g., `gini` or `entropy`.

1.2.2 Bootstrapping

The training data for each decision tree is generated through bootstrapping, which creates random samples with replacement. This ensures diversity among the trees.

Bootstrap Sample: Select n_{samples} random samples with replacement from the dataset.

1.2.3 Feature Selection

Random Forests utilize random feature selection to improve diversity. The number of features is determined by the `max_features` parameter:

- `sqrt`: Select $\sqrt{n_{\text{features}}}$ features.
- `log2`: Select $\log_2(n_{\text{features}})$ features.
- Integer value: Select a fixed number of features.

Features are chosen randomly without replacement from the total feature set.

1.2.4 Training the Forest

The training process involves:

1. Generating a bootstrapped sample of the training data for each tree.
2. Selecting a subset of features randomly.
3. Training a decision tree on the bootstrapped data using the selected features.
4. Storing each trained tree and its associated feature subset.

1.2.5 Prediction

The prediction phase involves aggregating the outputs of individual decision trees using majority voting:

$$\text{Final Prediction: } \arg\max_c \sum_{t=1}^{n_{\text{trees}}} \mathbb{I}(h_t(x) = c),$$

where $h_t(x)$ is the prediction of tree t for instance x, and $\mathbb{I}$ is the indicator function.

2 Hyperparameter settings

A grid-search optimization algorithm is used to find the optimal choice of hyperparameters H_i out of some discrete selections of values:

$$\begin{gathered}\min_H \texttt{cost}(y, (\texttt{model}(H))(x)) \\ s.t. H = \{H_1, \ldots, H_k\}, \\ H_i \in \{h_{i1}, \ldots, h_{im_i}\}\end{gathered}$$

For simplicity, the custom SVR implementation fixes the kernel to be a second degree polynomial: $k(x, y) = (x^t y + 0.1)^2$. The moving hyperparameters are $\epsilon \in \{10^-1, 10^-2\}$ and $C \in \{0.1, 0.5, 1.0\}$. For the QP version, the grid search optimization algorithm found that optimal results are achieved when $C = 0.1, \epsilon = 0.1$. For the SMO version, the hyperparameters were tuned to $C = 0.1, epsilon = 0.01$

For Random Forest, the hyperparameters are configured to include the splitting criterion (`criterion`) set to either `gini` or `entropy`, the number of estimators (`n_estimators`) chosen from {2, 10, 20}, the maximum depth of the trees (`max_depth`) set to values in {3, 5}, and the maximum number of features considered for splitting (`max_features`) selected from {5, 10}. The grid search optimization algorithm determined that the optimal hyperparameters are achieved when `criterion` is `gini`, `n_estimators` is 20, `max_depth` is 5, and `max_features` is 5.

3 Evaluation metrics

The regression task is evaluated using the following metrics:

- **Mean Absolute Error** measures the errors between the predicted values and ground truth ovservations:
$$MAE(y_t, y_p) = \sum_{i=1}^{n} |y_i^t - y_i^p|$$
;
- **Root Mean Squared Error** represent the standard deviation of the error:
$$RMSE(y_t, y_p) = \sqrt{\sum_{i=1}^{n} \frac{(y_i^t - y_i^p)^2}{n}}$$
;
- **Normalized Root Mean Squared Error** is a standardized variant of RMSE that makes the metric more comaprable accross different data applications:
$$NRMSE(y_t, y_p) = \sqrt{\frac{\sum_{i=1}^{n}(y_i^t - y_i^p)^2}{\sum_{i=1}^{n}(y_i^t)^2}}$$
;
- **R2 score** is a statistical metric that determines the proportion of variance that can be explained by the independent variable:
$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i^t - y_i^p)^2}{\sum_{i=1}^{n}(y_i^t - \texttt{mean}(y^t))^2}$$
.

The classification task is evaluated using the following metrics:

- **Accuracy** measures the proportion of correctly classified instances among the total instances:
$$\text{Accuracy}(y_{\text{true}}, y_{\text{pred}}) = \frac{TP + TN}{TP + TN + FP + FN},$$
where TP, FP, TN, and FN represent the counts of true positives, false positives, true negatives, and false negatives, respectively.
- **Precision** quantifies the proportion of true positive predictions among all positive predictions:
$$\text{Precision}(y_{\text{true}}, y_{\text{pred}}) = \frac{TP}{TP + FP}.$$

- **Recall** (also known as Sensitivity) measures the proportion of actual positives that are correctly identified:

$$\text{Recall}(y_{\text{true}}, y_{\text{pred}}) = \frac{TP}{TP + FN + \epsilon},$$

where ϵ is a small constant added to avoid division by zero.

- **F-Measure** (or F1 Score) represents the harmonic mean of precision and recall:

$$F_1(y_{\text{true}}, y_{\text{pred}}) = \frac{2 \cdot TP}{2 \cdot TP + FP + FN + \epsilon},$$

where ϵ is a small constant added to avoid division by zero.

4 Experimental results

The SVR models are evaluated using 10-fold cross-validation. The entire dataset is split into 10 disjoint subsets. At one step, a subset is selected as the validation set and the other 9 subsets are joined into a training set. The model is trained on this train set and validated on the leftover set. Then, another subset is chosen for validation. Metrics are recorded for each step and statistically analyzed by their mean, standard deviation and confidence intervals. Tables 1 and 2 showcase experimental results for SVR, both QP and SMO implementations.

Metric	Mean	Std	Min	Max	C.I. (95%)	C.I. (50%)
MAE	1.820e12	4.643e12	5.084e9	1.569e13	(-1.681e12, 5.322e12)	(7.329e11, 2.908e12)
RMSE	2.763e12	7.239e12	7.589e9	2.442e13	(-2.695e12, 8.222e12)	(1.067e12, 4.459e12)
NRMSE	4.648e12	1.224e13	1.264e10	4.128e13	(-4.581e12, 1.387e13)	(1.780e12, 7.515e12)
R2	-2.638e26	7.861e26	-2.622e27	-2.499e20	(-8.566e26, 3.289e26)	(-4.480e26, -7.970e25)

Table 1: SVR Metrics (QP version)

Metric	Mean	Std	Min	Max	C.I. (95%)	C.I. (50%)
MAE	1.384e8	1.742e8	2.658e7	5.959e8	(7.061e6, 2.698e8)	(9.765e7, 1.792e8)
RMSE	1.742e8	2.123e8	3.458e7	7.265e8	(1.411e7, 3.343e8)	(1.244e8, 2.239e8)
NRMSE	2.903e8	3.488e8	5.685e7	1.194e9	(2.736e7, 5.533e8)	(2.086e8, 3.720e8)
R2	-3.252e17	6.772e17	-2.264e18	-5.131e15	(-8.358e17, 1.854e17)	(-4.838e17, -1.666e17)

Table 2: SVR Metrics (SMO version)

Metric	Mean	Std	Min	Max	C.I. (95%)	C.I. (50%)
MAE	0.1616	0.0390	0.1112	0.2323	(0.1322, 0.1910)	(0.1524, 0.1707))
RMSE	0.3175	0.1043	0.1417	0.4929	(0.2389, 0.3962)	(0.2931, 0.3420)
NRMSE	0.3175	0.1043	0.1417	0.4929	(0.2389, 0.3962)	(0.2931, 0.3420)
R2	0.8777	0.0719	0.7494	0.9765	(0.8235, 0.9319)	(0.8609, 0.8945)

Table 3: SVR Metrics (SMO version, C=1.0, ϵ-0.1)

The Random Forest classifier implemented from scratch and the one used from the `sckikit-learn` library [PVG+11] are also evaluated using 10-fold cross-validation. Tables **??** and **??** showcase experimental results for both the Random Forest classifiers.

Metric	Model	Mean	Std	C.I. (95%)
Accuracy	Model from Scratch	0.998	0.0040	(0.9949, 1.0010)
	Model from Library	0.9970	0.0046	(0.9935, 1.0005)
Precision	Model from Scratch	0.9942	0.0116	(0.9855, 1.0030)
	Model from Library	0.9915	0.0130	(0.9818, 1.0013)
Recall	Model from Scratch	1.0000	0.0000	(1.0000, 1.0000)
	Model from Library	1.0000	0.0000	(1.0000, 1.0000)
F-Measure	Model from Scratch	0.9971	0.0059	(0.9927, 1.0015)
	Model from Library	0.9957	0.0066	(0.9907, 1.0007)
Execution Time (s)	Model from Scratch	1.3118	0.2635	(1.1131, 1.5104)
	Model from Library	0.0112	0.0003	(0.0110, 0.0114)

Table 4: Comparison of Classification Metrics for the RF from Scratch and RF from `sckit-learn` library.

5 Discussion

5.1 SVR

The custom implementations have poor performance due to the weak optimization solvers related to the high dimensionality of the dataset. It can still be noted that the SMO algorithm dedicated to the SVR problem still performs orders of magnitude better than the generic Newton smoothing method. The model keeps its robustness among cross-validation steps, with no huge relatively huge variation when train and validation sets are changed (e.g. for a certain train-val setup, the model performs considerably better than for other sets).

SHapley Additive exPlanations (SHAP) is a game-based method to explain the output of the model using SHAP values, which estimate the contribution of each feature to the prediction.

The high errors in the custom SVR models causes SHAP to fail produce meaningful values (as seen in Figures 1 and 2). For comparison, a SHAP bar plot on the library implementation (Figure 3), reveals that features like date, CO2, sound and some light sensor measurements have a positive impact on the final prediction. The CO2 and light values tend to have the highest impact on the final prediction. The other features altogether contribute negatively.

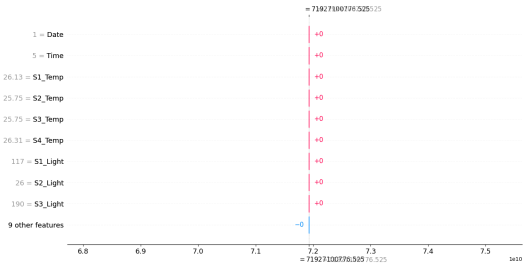

Figure 1: Shap plot for SVR (QP version)

The literature models use the room occupancy dataset for classification tasks. To facilitate comparison, the predicted values from SVR regression are converted to labels by applying a threshold to continuous outputs (1 if output is positive, -1 if negative). The results can be consulted in 5. The authors of the dataset applied a handful of machine learning methods on this data, among which the SVM with linear and RBF kernel belong to the same family as SVR. Table 5 show that the performance of custom implementation is almost close to random guessing, while sklearn's SVR achieves satisfying results, but still weaker than the author's SVM. Additionally, sklearn versions of SVM classifiers were trained and fine-tuned in the same way as the SVR for linear ($C = 0.5$) and rbf ($C = 0.1$) kernels. The results,

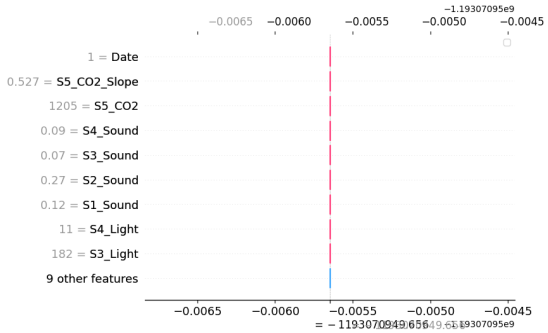

Figure 2: Shap plot for SVR (SMO version)

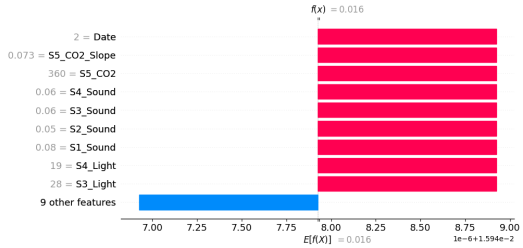

Figure 3: Shap plot for SVR (sklearn)

visible in table 6, surpass the SVR performance, and the linear SVC is the best model tested so far.

Metric	SVR SMO	SVR sklearn	SVM lin. [SJC+18]	SVM RBF [SJC+18]
Accuracy	0.5959 ± 0.0370	0.861 ± 0.0069	0.982	0.984
F1	0.5204 ± 0.0281	0.794 ± 0.0081	0.948	0.953

Table 5: Comparison with state-of-the-art and library implementation

5.2 Random Forest

The comparison of metrics between the Random Forest (RF) model implemented from scratch and the RF model from the `scikit-learn` library demonstrates several insights into their relative performance and computational efficiency.

The RF model from scratch achieves slightly higher metrics for **Accuracy** (0.998 vs. 0.997), **Precision** (0.9942 vs. 0.9915), and **F-Measure** (0.9971 vs. 0.9957), with both models achieving a perfect **Recall** (1.000). This indicates that the custom implementation can produce competitive predictions, but the differences in metrics are negligible and fall within the confidence intervals.

However, the **Execution Time** reveals a significant drawback of the custom implementation, taking an average of 1.3118 seconds per run compared to only 0.0112 seconds for the library version. This disparity highlights the substantial computational overhead associated with the scratch implementation due to less optimized internal structures and algorithms.

The custom implementation demonstrates robustness in terms of accuracy and precision but suffers from inefficiencies in execution time. These results underline the efficiency and scalability advantages of well-maintained libraries like `scikit-learn`, particularly for real-world applications requiring rapid and repeated predictions.

The SHAP plots (Figure 4 and 5) reveal insights into the contributions of individual features to the model's predictions. In both plots, the `Date` feature emerges as the most

Metric	SVC lin	SVC rbf
Accuracy	0.9980 ± 0.0006	0.9600 ± 0.0030
F1	0.9970 ± 0.0008	0.941 ± 0.0051

Table 6: Results of locally trained SVC classifiers that match Singh et al.'s setup [SJC+18]

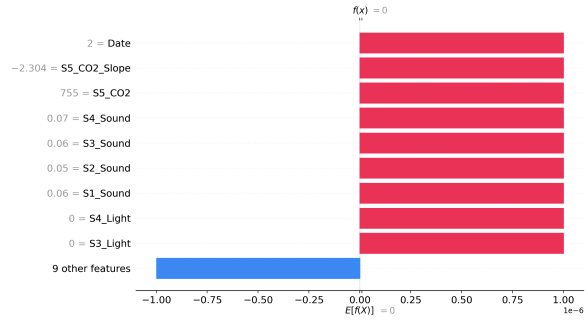

Figure 4: SHAP plot for Random Tree Classification (from scratch)

significant driver of the model's output, with a consistently strong positive contribution. This indicates that temporal information embedded in the `Date` variable plays a crucial role in the decision-making process of the model.

Another key feature is `S5_CO2_Slope`, which has a pronounced negative contribution in both plots. This suggests that this feature inversely affects the model's predictions, highlighting its importance as a counterbalance to the positive influences of other features. The combination of these two dominant features (`Date` and `S5_CO2_Slope`) demonstrates that the model relies heavily on temporal and CO2-related measurements for its predictions.

The `S5_CO2` feature also consistently contributes positively, albeit with less impact than the `Date` and `S5_CO2_Slope` features. Additionally, the sound-related features (`S4_Sound`, `S3_Sound`, `S2_Sound`, `S1_Sound`) provide smaller, yet consistent, positive contributions. These features suggest that acoustic signals may provide supplemental context for the model's predictions.

Conversely, the light-related features (`S4_Light` and `S3_Light`) and the collective grouping of "9 other features" exhibit negligible contributions. Their limited impact implies that these variables do not provide meaningful information to the model in its current configuration and task.

Across both plots, the consistency of feature importance reinforces the reliability of the SHAP explanations. The model's reliance on a few dominant features, complemented by the moderate influence of others, provides a clear picture of the feature hierarchy. While the model effectively leverages temporal, CO2, and sound-related features, the absence of meaningful contributions from light-related variables and others may suggest an opportunity to refine feature engineering or reassess their relevance to the predictive task.

Given these findings, the custom implementation serves as a valuable learning tool but is not practical for deployment in performance-critical environments. The library implementation, with its fine-tuned optimizations, remains the preferred choice for both speed and comparable accuracy.

References

[CL11] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

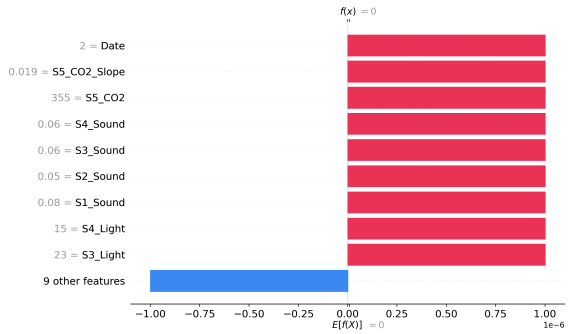

Figure 5: SHAP plot for Random Tree Classification (from `sckit-learn`)

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[SJC+18] Adarsh Pal Singh, Vivek Jain, Sachin Chaudhari, Frank Alexander Kraemer, Stefan Werner, and Vishal Garg. Machine learning-based occupancy estimation using multivariate sensor nodes. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2018.

[SS04] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199 – 222, 2004. Cited by: 8991.