Machine Learning - Software Project

Component 3

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

1 Target function

The problem that we are covering in this software project is to determine whether there are or not people in a room based on 24 sensor measurements and the timestamps when the measurements were taken. Because the timestamps were modelled into 2 numbers in the preprocessing step (the date as a class between the dates in the dataset and the time as a class representing the time range) and there were 22 numerical values measured by the sensors for each data point, we can define the domain of the target function as being $\mathbb{R}^{24}$, with an example of an input:

$x =$ { Date, Time, S1_Temp, S2_Temp, S3_Temp, S4_Temp,
S1_Light, S2_Light, S3_Light, S4_Light, S1_Sound, S2_Sound,
S3_Sound, S4_Sound, S5_CO2, S5_CO2_Slope, S6_PIR, S7_PIR }.

The co-domain of the target function will depend on the learning task:

- For **supervised classification**, the co-domain of the target function will be a class representing that either there are persons in the room (1) or there are not (0). So the co-domain of the target function will be $\{0, 1\}$, and the target function will be defined as:

$$f_{\text{classification}} : \mathbb{R}^{24} \to \{0, 1\}$$

The conclusion of the model in this case will be given by the output of the function:

$$f_{\text{classification}}(x) = 0 \Rightarrow \text{there is noone in the room}$$

$$f_{\text{classification}}(x) = 1 \Rightarrow \text{there is at least 1 person in the room}$$

- For **supervised regression**, the output of the target function will be a real number, so the co-domain of the function is $\mathbb{R}$ and the target function will be defined as:

$$f_{\text{regression}} : \mathbb{R}^{24} \to \mathbb{R}$$

The conclusion of the model in this case can be extracted in multiple ways, in our case we will conclude if there are either people in the room or not in the following way:

$$f_{\text{regression}}(x) < 0 \Rightarrow \text{there is noone in the room}$$

$$f_{\text{regression}}(x) \geq 0 \Rightarrow \text{there is at least 1 person in the room}$$

2 Support Vector Regression

Support Vector Regression (SVR) is an eager supervised machine learning algorithm designed for regression problems [GGNS21]. It tries to predict a continuous value from a set of discrete observations. In this regard, SVR extends the classical binary classification model of Support Vector Machines (SVM) in order to make it suitable for regression. It introduces an ϵ-insensitive region around the function (in the linear case, a separation hyperplane), which is called the ϵ-tube. The approximated function is obtained as the solution of an optimization problem that tries to find the narrrowest tube that optimizes the prediction error.

2.1 Representation of the learning function

The goal of SVR is to learn a function $f : \mathbb{R}^d \to \mathbb{R},\ f(x) = w^t \cdot \phi(x) + \gamma$ that approximates the continuous target y for each sample x with a certain degree of tolerance ϵ [GGNS21], where ϕ is a (nonlinear) mapping function to higher dimensions as found in SVM formulations [CL11]. The case when a certain function does not exist is covered by introducing two variables $\xi^+, \xi^- \geq 0$ signifying the positive and negative deviations from the wanted tolerance [GGNS21]. They are computed using an ϵ-sensitive loss

$$\|\xi\|_\epsilon = \max(0, \|\xi - \epsilon\|) \text{ [SS04]}.$$

Keeping these variables as low as possible make the subject for optimization, along with the weights as a regularization parameter. Formally, the SVR optimization problem can be written as follows:

$$\begin{aligned} \min\ & \|w\|_2^2 + C \cdot \sum_{i=1}^{n} (\xi_i^+ + \xi_i^-) \\ \text{subject to}\quad & y_i - w^t \phi(x_i) - \gamma \leq \epsilon + \xi_i^+, \\ & w^t \phi(x_i) + \gamma - y_i \leq \epsilon + \xi_i^-, \\ & w \in \mathbb{R}^d, \gamma \in \mathbb{R}, \\ & \xi_i^+, \xi_i^- \geq 0\ \forall\ i = \overline{1,n}, \end{aligned}$$

where C is a hyperparameter that controls the impact of the deviation from the tolerance ϵ [GGNS21].

2.2 Learning hypothesis

The hypothesis is the tube around the hyperplane defined by weights w and the tolerance ϵ.

This is a quadratic optimization problem. With the help of Lagrange multipliers and Karush-Kuhn-Tucker conditions [SS04], the dual problem is defined as

$$\begin{aligned} \min_{\alpha^-, \alpha^+}\ & \frac{1}{2}(\alpha^- - \alpha^+)^t Q (\alpha^- - \alpha^+) + \epsilon \sum_{i=1}^{n} (\alpha_i^- + \alpha_i^+) + \sum_{i=1}^{n} y_i (\alpha_i^- - \alpha_i^+) \\ \text{subject to}\quad & \textstyle\sum_{i=1}^{n} (\alpha_i^- - \alpha_i^+) = 0, \\ & 0 \leq \alpha_i^-, \alpha_i^+ \leq C, \forall\ i = \overline{1,n}, \end{aligned}$$

where $Q_{ij} = K(x_i, x_j) = \phi(x_i)^t \phi(x_j)$ is the value of the kernel function applied in data points x_i and x_j [CL11].

The representation of the learned hypothesis h is therefore

$$h(x) = \sum_{i=1}^{n} (\alpha^+ - \alpha^-) K(x_i, x) + \gamma$$

Applying the regression alorithm to a classification algorithm implies that the hypothesis should satisfy the conditions: $h(x) < -\epsilon$ if $f(x) = 0$ and $h(x) > \epsilon$ if $f(x) = 1$.

A polynomial was chosen to be the kernel of the SVR machine: $K(x_i, x_j) = (x_i^t x_j + c)^2$.

2.3 Learning algorithm [CL11]

Rewrite the dual problem (D):

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

$$\text{subject to} \quad u^t \begin{bmatrix} \alpha^+ \\ \alpha^- \end{bmatrix} = 0, \ 0 \leq \alpha_i^-, \alpha_i^+ \leq C, \forall \ i = \overline{1, n}.$$

,

where $u = \begin{bmatrix} e^t & -e^t \end{bmatrix}^t, e = [1 \ 1 \ \dots \ 1]^t \in \mathbb{R}^n$.

In the following, let $\alpha = \begin{bmatrix} (\alpha^+)^t & (\alpha^-)^t \end{bmatrix}^t$ be the cumulated variable of the problem, and $p = \begin{bmatrix} \epsilon e^t - y^t, \epsilon e^t + y^t \end{bmatrix}$.

2.3.1 SMO-type decomposition algorithm

Algorithm $Solver(D) \rightarrow \alpha_{opt}$

Preconditions: (D) is the quadratic optimzation problem in the form defined above

Postconditions: α_{opt} holds the optimal solution of (D)

1. Find an initial feasible solution α^1 (e.g. $\alpha^1 = 0$ works). Set $k = 1$.

2. **If** a^k is a stationary point of (D), **then** $\alpha_{opt} \leftarrow \alpha^k$ **stop**.
   Else, apply $WSS1(\alpha^k)$ to obtain a two element working set $B = \{i, j\}$. Let $N = \{1, \dots, n\} \setminus B$, and α_B^k, α_N^k be the subvectors of α with indices in B, respectively N.

3. **If** $a_{ij} = K(x_i, x_i) + K(x_j, x_j) - 2K(x_i, x_j) > 0$,
   then solve the following subproblem (E) with variable $\alpha_B = [\alpha_i \ \alpha_j]^t$:

$$\min_{\alpha_B} \alpha_B^t \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj} \end{bmatrix} \alpha_B + (p_B + Q_{BN} \alpha_N^k)^t \alpha_B$$

$$\text{subject to} \quad u_B^t \alpha_B = 0 - u_N^t \alpha_N^k,$$

   else let τ be a small positive constant and solve (F):

$$\min_{\alpha_B} \alpha_B^t \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj} \end{bmatrix} \alpha_B + (p_B + Q_{BN} \alpha_N^k)^t \alpha_B + \frac{\tau - a_{ij}}{4} ((\alpha_i - \alpha_i^k)^2 + (\alpha_j - \alpha_j^k)^2)$$

$$\text{subject to} \quad u_B^t \alpha_B = 0 - u_N^t \alpha_N^k,$$

4. Set α_B^{k+1} to be the optimal solution to the problem (E) or (F) and $\alpha_N^{k+1} = \alpha_N^k$.

5. $k \leftarrow k + 1$ and go to step 2.

2.3.2 WSS 1 algorithm

For a solution α define the sets of indices

$$I_{up}(\alpha) = \{t | \alpha_t < C, u_t = 1 \ \vee \ \alpha_t > 0, y_t = -1\}$$

$$I_{low}(\alpha) = \{t | \alpha_t < C, u_t = -1 \ \vee \ \alpha_t > 0, y_t = 1\}$$

Algorithm $WSS1(\alpha^k) \to B$
Preconditions: α^k partial solution of (D) that is not a stationary point
Postconditions: B is a set of two indices from the variable vector α^k that are used to formulate the optimization subproblems.
Pseudocode:

1. **For** all $t, s = (\overline{1,n})^2$ **define**
   $a_{ts} \leftarrow K(x_t, x_t) + K(x_s, x_s) - 2K(x_t, x_s)$
   $b_{ts} \leftarrow -u_t \nabla_t f(\alpha^k) + u_s \nabla_s f(\alpha^k)$
   $\overline{a}_{ts} \leftarrow a_t s$ `if` $a_t s > 0$ `else` τ

2. **Select**
   $i \in \arg\max_t \{-u_t \nabla_t f(\alpha^k) | t \in I_{up}(\alpha^k)\}$
   $j \in \arg\min_t \{-b_{it}/\overline{a_{it}} | t \in I_{low}(\alpha^k), -u_t \nabla_t f(\alpha^k) < -u_i \nabla_i f(\alpha^k)\}$

3. **Return** $B = \{i, j\}$.

4. **Stop**.

2.3.3 Finding γ

If for a solution α there exists at least component α_i such that $0 < \alpha_i < C$, take

$$\gamma = \texttt{mean}_i(-u_i \nabla_i f(\alpha)).$$

Otherwise, b can be chosen from a bounded interval

$$m(\alpha) \leq \gamma \leq M(\alpha),$$

where

$$m(\alpha) = \min\{u_i \nabla_i f(\alpha) | \alpha_i = 0, u_i = 1 \vee \alpha_i = C, u_i = -1\},$$

$$M(\alpha) = \max\{u_i \nabla_i f(\alpha) | \alpha_i = 0, u_i = -1 \vee \alpha_i = C, u_i = 1\}.$$

An example of choice can be the middle point of the interval,

$$\gamma = \frac{m(\alpha) + M(\alpha)}{2} \text{ [CL11]}.$$

3 Random Forest Classification

Random Forest (RF) is a powerful ensemble-based machine learning algorithm used for both classification and regression tasks. In classification, RF builds a collection of the Decision Trees (DTs) week learners and combines their outputs to improve prediction accuracy and reduce overfitting [Bre01]. By aggregating the results of multiple DTs, RF leverages the diversity among them to achieve robust and accurate predictions [Zho12].

3.1 Representation of the Learning Algorithm

Random Forest is constructed as an bagging ensemble of T decision trees $\{h_t\}_{t=1}^{T}$, where each tree h_t is trained on a different bootstrap sample of the training data $D = \{(x_i, y_i)\}_{i=1}^{n}$ and $y_i \in \{1, 2, \ldots, K\}$ represents the class labels for a K-class classification problem [Bre01]. To introduce additional randomness, at each split in a tree, only a randomly chosen subset of m features is considered.

The predicted class for an input sample x is determined by majority voting among the T trees:

$$\hat{y} = \arg \max_{k \in \{1,\ldots,K\}} \sum_{t=1}^{T} \mathbb{I}[h_t(x) = k],$$

where $\mathbb{I}[h_t(x) = k]$ is an indicator function that equals 1 if tree h_t assigns class k to x, and 0 otherwise.

The goal of each DT is to learn a set of decision rules at each node (e.g., "if $x_i >$ 0.5go to the left sub-tree, else go to the right sub-tree that can classify an input ($x \in R^k$) to some class (the leaves of the tree). This model can then be used for making predictions by following the decision rules from the root to a leaf node for new input data.

3.2 Learning Hypothesis

The learning hypothesis of each Decision Tree which are composing the Random Forest is the Decision Tree itself (so what decisions should be done at each step) which will be learning during the training phase (so this model learns eagerly). The learning hypothesis of the Random Forest tree, on the other hand, is the ensemble of all the Decision Trees within the forest. This means that the Random Forest learns by aggregating the predictions of multiple individual Decision Trees, typically through majority voting for classification tasks.

3.3 Decision Tree Learning Algorithm [Bre17]

A decision tree is a supervised machine learning algorithm used for both classification and regression tasks. It learns to predict target values by recursively splitting the input data based on feature thresholds, aiming to maximize the separation of target classes or minimize prediction error [Bre17].

This model is structured as a binary tree where each internal node represents a decision rule on a feature, and each leaf node represents a prediction (e.g., a class label or a numerical value). The learning process involves building this tree iteratively from the root to the leaves by choosing the most discriminative feature splits.

The algorithm typically follows these steps:

1. **Initialization:** Start with the entire dataset at the root node.
2. **Splitting Criterion:** For each possible feature and threshold, compute the effectiveness of the split using a criterion such as:
   - **Gini Impurity:** Measures the likelihood of incorrectly classifying a randomly chosen element:
     $$Gini = 1 - \sum_{k=1}^{K} p_k^2,$$
     where p_k is the proportion of class k in the node.

- **Entropy:** Measures the information gain achieved by a split:

$$Entropy = -\sum_{k=1}^{K} p_k \log_2(p_k).$$

3. **Split Selection:** Choose the feature and threshold that minimize impurity or maximize information gain. Partition the data into left and right child nodes accordingly.

4. **Recursion:** Repeat steps 2 and 3 for each child node until a stopping condition is met, such as:

   - A maximum tree depth is reached.
   - The number of samples in a node falls below a specified threshold.
   - Further splits do not result in a significant reduction in impurity.

3.4 Random Forest Learning Algorithm [Bre01]

Algorithm: $RandomForestClassifier(D, T, m) \rightarrow H$

Preconditions:

- $D = \{(x_i, y_i)\}_{i=1}^{n}$ is the labeled training dataset.
- T is the number of decision trees to be trained.
- m is the number of randomly selected features considered at each split.

Postconditions:

- $H = \{h_t\}_{t=1}^{T}$ is the ensemble of decision trees.

Pseudocode:

1. **For** $t = 1$ to T:
   (a) Sample a bootstrap dataset D_t of size n by randomly selecting data points from D with replacement.
   (b) Train a decision tree h_t on D_t using the algorithm presented in Subsection 3.3.
2. **Return** the ensemble of trees $H = \{h_t\}_{t=1}^{T}$.

3.5 Prediction

To predict the class of a new sample x, the following steps are applied:

- Pass x through all T decision trees in H to obtain their individual predictions $\{h_t(x)\}_{t=1}^{T}$.
- Compute the class-wise vote count:

$$v_k = \sum_{t=1}^{T} \mathbb{I}[h_t(x) = k], \quad \forall k \in \{1, \ldots, K\}.$$

- Assign x to the class with the highest vote count:

$$\hat{y} = \arg \max_{k \in \{1,\ldots,K\}} v_k.$$

3.6 Hyperparameters

Random Forest has several important hyperparameters that influence its performance:

- **Number of Trees (T):** Controls the size of the ensemble.
- **Number of Features (m):** Determines the randomness of feature selection at each split.
- **Maximum Depth:** Limits the depth of each decision tree to prevent overfitting.
- **Minimum Samples per Leaf:** Sets the minimum number of samples required to form a leaf node.

References

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[Bre17] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[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.

[GGNS21] Claudio Gambella, Bissan Ghaddar, and Joe Naoum-Sawaya. Optimization problems for machine learning: A survey. *European Journal of Operational Research*, 290(3):807–828, 2021.

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

[Zho12] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.