Machine learning models mathematical notations and hyperparameter tuning

Six machine learning models obtained from the scikit-learn library [1] were carefully implemented during the analysis conducted for this study. The scikit-learn documentation covers the theory behind these models and how to implement them. This document summarizes the fundamental mathematical equations most important to our studies. Furthermore, hyperparameter tuning was performed, wherein particular adjustments were made to the ANN architecture using multi-layer perception, the kernel selection for support vector machines (SVM), and the regularization in logistic regression (LR).

For the remaining hyper-parameters, a systematic optimization procedure was carried out using GridSearchCV. This method ensured that every model was tailored to suit the specified parameter values by conducting a comprehensive search. The refined selection of hyper-parameters through GridSearchCV greatly helped enhance the predictive performance and ensure the models operated at an optimal level within the context of this study.

1 Naive bayes

In this study, the naive bayes (NB) model assumes a Gaussian distribution for the likelihood of the features. The conditional probability for a feature x_i given a class y is modeled as:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

where σ_y and μ_y are the standard deviation and mean of the features for class y, estimated using maximum likelihood.

2 Random forests

In this analysis, the random forest (RF) model is employed, an ensemble that synthesizes multiple decision trees to establish a robust prediction mechanism. The significance of features is inferred from the decrement in node impurity, a metric weighted by the probability of reaching each node. Employing the Gini Index as an impurity measure, Scikit-learn assesses each node's importance within the tree. The importance ni_j of node j, is determined as follows:

$$ni_j = w_j C_j - w_{\text{left}(j)} C_{\text{left}(j)} - w_{\text{right}(j)} C_{\text{right}(j)}$$

where, w_j signifies the weighted number of samples arriving at node j, C_j denotes the node's impurity, and left(j) and right(j) represent the left and right child nodes originating from node j, respectively.

For each attribute within a single tree, significance is measured by the ratio of its weighted node importance to the sum of all nodes' importances, computed as:

$$f_i = \frac{\sum_{j:\text{node } j \text{ splits on feature } i} n_{ij}}{\sum_{k \text{ in all nodes}} n_{ik}}$$

where f_i denotes the significance of the i-th attribute and n_{ij} the significance of the j-th node. This measure is then standardized across the interval [0, 1] through normalization against the sum of all attribute significances:

$$\text{norm}f_i = \frac{f_i}{\sum_{j \text{ in all features}} f_j}$$

At the level of the RF, the consolidated significance of a feature is the average significance computed across all the trees:

$$RFf_i = \frac{\sum_{j \text{ in all trees}} \text{norm}f_{ij}}{T}$$

where RFf_i symbolizes the composite significance of the i-th feature across the entire RF, $\text{norm}f_{ij}$ the normalized significance of the i-th feature in the j-th tree, and T the total number of trees in the forest, set to a maximum of 4000 iterations for the purpose of this study.

3 Decision tree

This study applied a decision tree (DT) algorithm to a binary classification task, utilizing the gini index as the impurity metric. This impurity measure is crucial in assessing the suitability of binary splits within the tree's structure. Binary splits were performed based on a threshold value established at 0.5, categorizing the samples into two distinct classes: positive and negative outcomes. The tree was constructed through an iterative process that sought to identify the optimal feature and threshold for splitting, aiming to achieve the greatest reduction in impurity. This iterative process was bounded by predefined stopping criteria, such as reaching a minimum sample size at a node, achieving the maximum depth of the tree, or reaching the set maximum number of iterations, which was 4000.

The Gini index, which quantifies node impurity, is mathematically expressed as:

$$H(Q_m) = \sum_{k=1}^{K-1} p_{mk}(1 - p_{mk})$$

where p_{mk} denotes the proportion of observations of class k at node m. The algorithm minimizes this impurity criterion for all potential splits and thresholds.

Considering a dataset with 27 features, let $x_i \in \mathbb{R}^{27}$ represent the training vectors and $y \in \mathbb{R}^L$ represent the label vector. The DT algorithm partitions the feature space to group samples with the same label or similar target values. At any given node m in the tree, defined by Q_m with n_m samples, a split is considered on feature j with threshold t_m, bifurcating the data into the left and right subsets $Q_m^{left}(\theta)$ and $Q_m^{right}(\theta)$, respectively:

$$Q_m^{left}(\theta) = \{(x, y) \mid x_j \leq t_m\}$$
$$Q_m^{right}(\theta) = Q_m \setminus Q_m^{left}(\theta)$$

The impurity of a split at node m is assessed using an impurity or loss function $H(\cdot)$, where the optimal split θ^* is the one that minimizes the impurity:

$$\theta^* = \arg\min_{\theta} G(Q_m, \theta)$$
$$G(Q_m, \theta) = \frac{n_m^{left}}{n_m} H(Q_m^{left}(\theta)) + \frac{n_m^{right}}{n_m} H(Q_m^{right}(\theta))$$

The splitting process is performed recursively for subsets $Q_m^{left}(\theta^*)$ and $Q_m^{right}(\theta^*)$ until one of the stopping conditions is met.

For nodes targeting classification with outcomes in $\{0, 1, \ldots, K-1\}$, the proportion p_{mk} of observations belonging to class k within node m is defined by:

$$p_{mk} = \frac{1}{n_m} \sum_{x_i \in Q_m} I(y_i = k)$$

4 Logistic regression

The logistic regression (LR) model was utilized for the predictive analysis, with a dataset containing 27 attributes. The probability estimate $p(x)$ was constrained within the range [0,1] through the logistic function, defined as:

$$\log\left(\frac{p(x)}{1-p(x)}\right) = \alpha_0 + \alpha \cdot x$$
$$p(x) = \frac{e^{\alpha_0 + \alpha \cdot x}}{1 + e^{\alpha_0 + \alpha \cdot x}}$$

where α represents the model's parameters. The LR model was calibrated to classify observations by setting a threshold at 0.5.

To fit the LR model to the data, the likelihood was maximized, capturing the probabilities of the observed outcomes. For each data point i, the likelihood is given by:

$$L(\alpha_0, \alpha) = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}$$

Taking the logarithm transforms the product into a sum, simplifying the calculation:

$$l(\alpha_0, \alpha) = \sum_{i=0}^{n} y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i))$$

Maximizing this log-likelihood function with respect to the parameters, using the 'newton-cg' solver, yields the model that best fits the data. The method of Maximum Likelihood Estimation was applied with a regularization strength C of 1 and a maximum iteration limit of 4000.

$$l(\alpha, x) = \sum_{i=0}^{n} -\log(1 + e^{\alpha_0 + \alpha \cdot x_i}) + \sum_{i=0}^{n} y_i(\alpha_0 + \alpha \cdot x_i)$$

5 Artificial neural network

In the conducted study, the Multi-layer Perceptron was employed as an artificial neural network (ANN), which is a versatile algorithm capable of handling complex non-linear relationships within the data. The ANN consisted of an input layer, three hidden layers with 13 neurons each, and an output layer designed for binary classification. The activation function for the neurons was the hyperbolic tangent function, defined as:

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

The ANN was optimized using the adam optimizer, an extension of stochastic gradient descent, which is particularly effective for problems with large datasets and parameters. It adjusts the learning rate dynamically, leading to faster convergence.

For the binary classification task, the network was trained to minimize the average cross-entropy loss function, which for a dataset with binary targets is formulated as:

$$\text{Loss}(y, \hat{y}, W) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] + \frac{\alpha}{2n}||W||_2^2$$

where $\hat{y}$ is the predicted output, y is the true target, W denotes the weights of the network, and α represents L2 regularization to prevent overfitting. The network was trained for a maximum of 4000 iterations, without a preset random state, to ensure comprehensive learning across various initial conditions.

6 Support Vector Machine

For the support vector machine (SVM) model, the radial basis function (RBF) kernel was selected to handle the non-linearity in the data. The RBF kernel, a popular choice for SVM, transforms the input space into a higher-dimensional space where the data points can be linearly separated. The formulation of the SVM with an RBF kernel is given by:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b$$

where $K(x_i, x) = e^{-\gamma||x_i - x||^2}$ is the RBF kernel, α_i are the Lagrange multipliers, y_i are the class labels, and b is the bias term.

The hyperparameter C, which controls the trade-off between a smooth decision boundary and classifying training points correctly, was set to 1. A higher value of C would have led to a more complex model that might potentially overfit the training data, while a smaller value would have created a smoother decision boundary with a higher bias. With C set to 1, the model sought to balance model complexity and generalization to unseen data.

The SVM model was trained to find the optimal hyperplane that maximizes the margin between the different classes. The margin is defined by the distance between the decision boundary and the closest data points from either class, known as the support vectors. The optimization problem for the SVM can be expressed as:

$$\min_{\alpha} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{n} \alpha_i$$

subject to the constraints $0 \leq \alpha_i \leq C$ for all i, and $\sum_{i=1}^{n} \alpha_i y_i = 0$.

The SVM was tuned and evaluated to ensure the best performance on the given dataset, striving to achieve a balance between precision and recall.

References

[1] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12:2825–2830, 2011.