Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Bachelor's diploma thesis

in the field of study Mathematics and Data Analysis

Evaluating Machine Learning Algorithm Performance Across Datasets: A Study Using the Elo-Based Predictive Power Measure on OpenML

Bartosz Szymański

student record book number 327223

thesis supervisor

dr Katarzyna Woźnica

academic advisor

prof. dr hab. inż. Przemysław Biecek

WARSAW 2025

Abstract

Evaluating Machine Learning Algorithm Performance Across Datasets: A Study Using the Elo-Based Predictive Power Measure on OpenML

In recent years, we have witnessed extraordinarily rapid development in machine learning—new algorithms and advanced models have already found extensive applications in fields such as medicine and finance. With the growing number of available solutions, the problem of conducting fair and rigorous comparisons has become increasingly important. Although numerous model-evaluation metrics have been proposed in the literature, many suffer from significant limitations that hinder reliable assessment. To address this gap, the Elo-based Predicted Power metric—based on the well-established Elo ranking system—was introduced.

The aim of this thesis is a cross-sectional analysis of the quality of machine learning models using the EPP quality measure. For this purpose, datasets provided by the OpenML platform were employed, with the EPP metric used to assess model effectiveness. The use of previously computed results from models contributed by platform users allowed for significant savings in computational resources. This approach enabled a comparison of algorithmic efficiency but also facilitated a deeper understanding of the EPP metric's applications and shows its practical utility in one of the best-known databases for tabular datasets.

Keywords: machine learning, algorithm evaluation, EPP measure, OpenML

Streszczenie

Ocena wydajności algorytmów uczenia maszynowego w zbiorach danych: Badanie z wykorzystaniem miary mocy predykcyjnej opartej na Elo w OpenML

W ostatnich latach obserwujemy niezwykle dynamiczny rozwój uczenia maszynowego – coraz to nowe algorytmy i zaawansowane modele znalazły już szerokie zastosowanie w takich obszarach jak medycyna czy finanse. Wraz z rosnącą liczbą dostępnych rozwiązań coraz większe znaczenie zyskuje problem ich rzetelnego porównywania. Choć w literaturze zaproponowano liczne metryki oceny jakości modeli, wiele z nich ma istotne ograniczenia, które utrudniają ewaluację. Jako odpowiedź na tę lukę w badaniach wprowadzono miarę Elo-based Predicted Power, opartą na sprawdzonym systemie rankingowym Elo.

Celem niniejszej pracy jest przekrojowa analiza jakości modeli uczenia maszynowego z wykorzystaniem miary jakości opartej na systemie Elo – EPP. Do tego celu wykorzystano zestawy danych udostępnione przez platformę OpenML, a do oceny skuteczności modeli zastosowano metrykę EPP. Wykorzystanie wyników modeli dostarczonych przez użytkowników platformy pozwoliło na znaczne oszczędności zasobów obliczeniowych. Podejście to umożliwiło porównanie efektywności algorytmów, a także ułatwiło głębsze zrozumienie zastosowań metryki EPP i potwierdziło jej praktyczną użyteczność na jednej z najbardziej znanych baz danych zawierających tabelaryczne zbiory danych.

Słowa kluczowe: uczenie maszynowe, ewaluacja algorytmów, miara EPP, OpenML

Contents

1. Introduction

Machine learning is currently one of the fastest-growing fields in science. New advancements in this area are rapidly being implemented across various sectors, including healthcare, finance, industry, and marketing. The fast pace of technological development and the abundance of available solutions have led to the emergence of numerous algorithms, and in practice, solving specific problems often involves building a wide variety of different models.

As a result, one of the key challenges has become comparing models in a way that enables the effective selection of the most optimal one for a given task. Although popular performance metrics—such as accuracy, F1-score, or AUC—partially assist in evaluating models, they come with significant limitations. Among the most notable issues are difficulties in interpreting the differences between model results, the lack of clear procedures to assess the statistical significance of these differences, and the inability to directly compare results across different datasets.

To address these limitations, a new comparative metric—EPP (Elo-based Predictive Power)—has been proposed. It is designed using logistic regression, which allows it to effectively handle the challenges mentioned above.

The goal of this thesis is to conduct a large-scale comparative analysis of the performance of machine learning algorithms using the EPP metric, thus extending the analysis from [Gosiewska et al., 2022]. For this purpose, the OpenML platform was used—one of the most important open repositories of datasets. OpenML not only provides access to data, but also includes information about the performance of trained models, the hyperparameters used, and benchmarking configurations, enabling model comparisons in a consistent and structured manner. This enables a comprehensive analysis without the need to train and test models, significantly saving computational resources. Additionally, it simplifies the potential expansion of the study to include a greater number of models, datasets, libraries or algorithms.

The comparative analysis of models available on OpenML using the EPP metric allows for a deeper understanding of the relationships between machine learning algorithms and the datasets used. Thanks to the wide range of models and datasets available on the platform, it is possible to examine how different algorithms perform in various contexts and on different types of data. This comprehensive evaluation contributes to identifying strengths and weaknesses of specific

methods, ultimately guiding more informed decisions in algorithm selection and development.

2. Fundamental Concepts

The aim of this chapter is to introduce the basic definitions and concepts related to machine learning. These concepts lay the groundwork for more advanced discussions in later chapters.

2.1. Machine learning basic concepts

Machine learning is an interdisciplinary field that combines mathematics, artificial intelligence, and computer science. Its goal is to develop algorithms capable of learning from data and generalizing information, much like human reasoning, without the need for explicit programming. Machine learning is applied to various problems such as clustering, regression or classification. Clustering is the task of grouping data into so-called clusters, which are groups of similar observations. Unlike regression and classification, clustering does not involve a target variable, so in this case, there is usually no single correct way to group the observations, as the data can be divided in many different ways, several of which may be valid. In the case of regression and classification, machine learning models use data to identify relationships between features and target variables. This enables making predictions or discovering patterns in unseen, similar data. This issue can be described as a predictive problem.

Definition 2.1 (Prediction problem). Let $\mathcal{X} \subset \mathbb{R}^p$ denote the feature space from p-dimensional space, and $\mathcal{Y} \subset \mathbb{R}$ the label space. Assume that there exists an joint probability distribution $\mathcal{P} = (\mathcal{X}, \mathcal{Y})$. The prediction problem involves determining a rule that assigns to each new observation from the feature space $\mathcal{X}$ an appropriate label value in $\mathcal{Y}$, based on a finite sample drawn from the distribution $\mathcal{P}$.

Finite sample from $\mathcal{P}$ is typically stored in datasets.

Definition 2.2 (Dataset). *Dataset* is a matrix consisting of n rows, also referred to as observations or records, and p+1 columns, which include p predictor variables (features) $X_i = (x_{1,i}...x_{n,i}) \in \mathbb{R}^n$ for $i = 1, \dots, p$ and target variable $Y = (y_1, \dots, y_n) \in \mathbb{R}^n$. Let denote

$(X_1, \ldots, X_p)$ as X, then

$$D_{n\times(p+1)} = (X, Y). \tag{2.1}$$

We assume that the rows of the matrix $D_{n\times(p+1)}$ is an independent and identically distributed (i.i.d.) sample from the distribution $\mathcal{P} = (\mathcal{X}, \mathcal{Y})$. Additionally, we will adopt the notation for the observations, meaning $x_i = x_{i,1}...x_{x,p}$ for $i = 1, ..., n$

During the model building process, the dataset is divided usually into two parts – the training and the test dataset. The training set is used to build the model, while the test set is used to evaluate its effectiveness. This division allows for a much more reliable evaluation of the model, as the model makes predictions on observations it has not used for learning. Using the same data for training and evaluating the model would lead to an overly optimistic error estimate and excessive adaptation to the given information, resulting in what is known as overfitting. Overfitting happens when the model learns the training data too well, making it less effective at handling new, unseen data. These concepts are closely related to the classification problem, which is the main focus of this thesis.

Definition 2.3 (Classification problem). We are given a training dataset $D = (X, Y)$, where Y is a target variable with a discrete set of values. The classification problem consists in finding a function that, based on the dataset D, learns the relationships between the set of feature vectors X and the corresponding classes Y, and subsequently enables the correct assignment of classes to new, previously unseen instances $x' \in \mathcal{X}$.

Definition 2.4 (Classification model). A classification model (classifier) is a function $f : \mathcal{X} \to \mathcal{Y}$ that takes a data point, and maps it to the predicted class label for that observation based on previously seen data.

When observations come from two classes, the problem is referred to as binary classification. In these cases we often denote $\mathcal{Y} = \{0, 1\}$, $\mathcal{Y} = \{-1, 1\}$ or $\mathcal{Y} = \{+, -\}$. Some classifiers not only provide a predicted class label but also output the probability that a given instance belongs to each class. This probabilistic information can be especially valuable, as it allows for more nuanced decision-making, confidence estimation, and the ability to adjust classification thresholds based on specific application needs. Fraud detection and determining whether messages are spam are examples of (binary) classification. The previously mentioned regression differs from classification primarily in the target variable – in regression, we predict continuous numerical values.

Given the importance of classification in various applications, selecting an appropriate model is crucial. Different classification models employ distinct methodologies, from simple rule-based approaches to complex machine learning techniques.

2.2. Machine learning models

In the following section, we briefly describe the most popular models and techniques that we refer to in the later part of the thesis.

2.2.1. Classical machine learning models

- Linear regression

  Linear regression is an algorithm primarily designed for regression problems. It is based on the assumption of a linear relationship between the target variable Y and the predictors $X_1, \dots, X_p$, which means it can be expressed as

  $$Y = X\beta + \epsilon,$$

  where $\epsilon = (\epsilon_1, \dots, \epsilon_n)$ is a random vector representing the noise, with the assumptions that expected value $E[\epsilon_i] = 0$ and constant variance $Var[\epsilon_i] = \sigma^2$ for $i = 1, \dots, n$. Vector $\beta = (\beta_0, \dots, \beta_p)^T$ represents the coefficients estimated in this method, typically in a way that minimizes the sum of the squared differences between the actual values and the predicted values, usually using the least squares method. The linear regression model is typically used to solve regression problems. However, it can also be applied to classification tasks, although this approach involves challenges due to the lack of constraints limiting the target variable's values to a range corresponding to class labels, e.g. [0, 1].

- Logistic regression

  In the case where we want to perform binary classification using linear models, we can use logistic regression to model the probability

  $$P(Y = 1|X) = \frac{1}{1 + e^{-X^T\beta}},$$

  where the right-hand side expression represents the sigmoid function applied at the point $X^T\beta$. This function transforms value of $X^T\beta$ from $(-\infty, +\infty)$ to $[0, 1]$, thus providing the probability of belonging to a given class. Vector $\beta = (\beta_0, \dots, \beta_p)^T$ denotes the model coefficients, which are typically estimated using the method of maximum likelihood estimation (MLE) [Hastie et al., 2009] or stochastic gradient descent (SGD) [Goodfellow et al., 2016].

- Naive Bayes Classifier

  It is a probabilistic model based on Bayes' theorem, which assumes conditional independence of features given the class. That is why it is called "naive". For a feature set $X = (X_1, \dots, X_n)$ and a target variable Y, the model estimates the posterior probability using the Bayes theorem:

  $$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)}.$$

  Assuming that the features are conditionally independent given the class, we can express:

  $$P(X \mid Y) = \prod_{i=1}^{n} P(X_i \mid Y),$$

  which simplifies computations and allows for efficient parameter estimation from training data. Classification is performed by selecting the class $\hat{y}$ that maximizes the posterior probability:

  $$\hat{y} = \arg\max_{y} \prod_{i=1}^{n} P(X = x \mid Y = y)P(Y = y).$$

2.2.2. Decision models

Decision models represent one of the most important classes of machine learning algorithms, valued for their interpretability and flexibility, which contribute to their widespread practical use.

- Decision trees

  These models have a binary tree structure. The trees consist of a root, which contains all observations. At each subsequent node, based on the value of one of the features, the data is split into two subsets. This leads to further nodes, thus partitioning the space. The nodes resulting from the final split are called leaves, and they are the subset of observations for which we determine the final prediction. At each internal node, the algorithm selects decision rule - a feature and a threshold that best separates the data. The splits are typically made in such a way as to maximize the information gain. This measure quantify the information carried by X, or in other words, the uncertainty about the value of X. A common measure used for this purpose is entropy. There are many variations and implementations of decision trees available, such as Random Tree, J48, and Decision Stump.

  In Figure 2.1, the set of observations denoted by X carries an information value of Q_1. We want to perform a split and have two variables to choose from for the test. In both

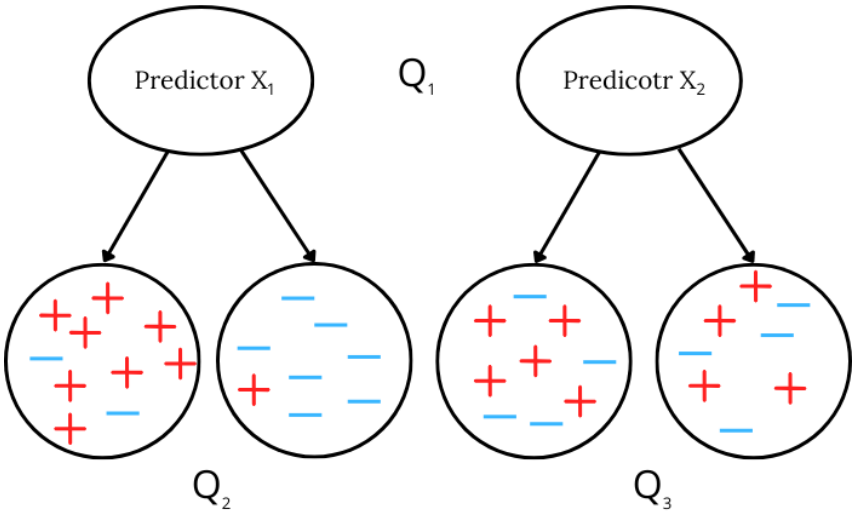

Figure 2.1: An example of choosing a decision rule. We see two potential splits based on different predictors, where plus signs represent observations of one class, minus signs represent observations of the other class, and Q_1, Q_2, Q_3 are the entropy values.

cases, the uncertainty about X will decrease, meaning $Q_1 > Q_2$, $Q_1 > Q_3$, because we have made a split of the observations. However, notice that the split based on predictor X_1 divides the space into significantly more homogeneous groups. This results in a lower entropy value for that split, making it a better choice.

- Rule based classifiers

  A rule-based classifier is a model that assigns a class label to an observation by evaluating a set of conditional rules. Each rule consists of a condition based on the input features, and if the condition is satisfied, the observation is assigned to a specific class. The conditions in the rules are typically logical expressions involving the feature values. Unlike decision trees, rule-based classifiers evaluate all rules independently and do not rely on a tree-like, hierarchically structure. This allows for more flexibility in combining conditions from different parts of the feature space, but can also make rule sets harder to interpret when they grow large. The class that is predicted by the most rules is chosen as the final output for the given input. An example of such a model is OneR. The OneR works by creating simple classification rules based on a single attribute – for each value of the attribute, it assigns the most frequently occurring class. Among all attributes, it selects the one with the lowest classification error and uses it to predict the class of new data.

2.2.3. Distance-based models

Another important group of models are distance-based models, which make decisions based on the distances between data points.

- Support vector machine (SVM)

  The goal of this model is to find the optimal hyperplane that divides the data. For linearly separable problems, model seeks the optimal hyperplane that maximizes its distance from the nearest points of each class, known as support vectors, effectively dividing the observations according to their respective classes. The minimum distance between an observation and the separating hyperplane is referred to as the margin. In the case of non-linearly separable problems, points may lie on the wrong side of the decision boundary, so then we introduce the penalty for observations that are misclassified or too close to the boundary. The model optimizes an objective function to find a balance between maximizing the margin and minimizing classification errors. In Figure 2.2, we see the output of this algorithm in a case with two classes in two dimensions. On the left, we have a linearly separable problem, and on the right, a non-linearly separable problem.

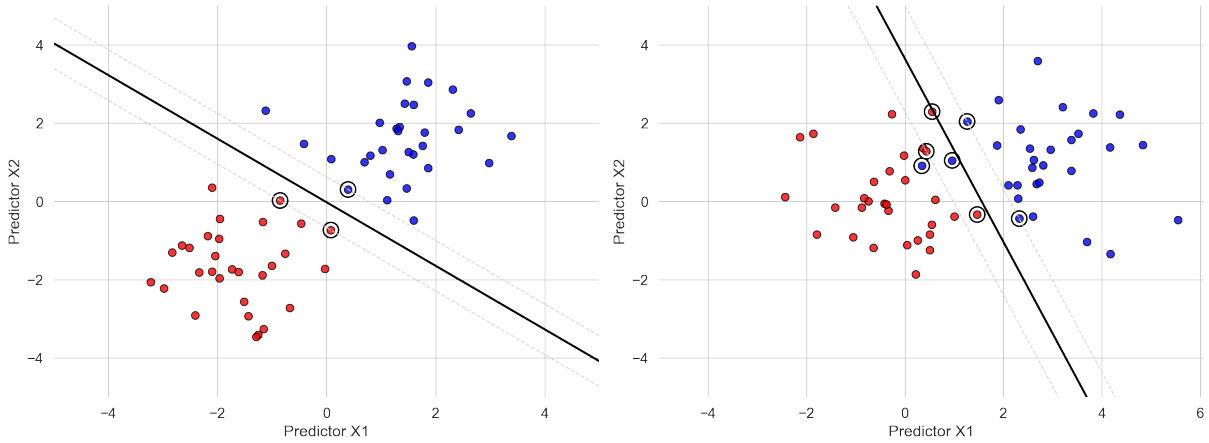

Figure 2.2: Examples of applying the SVM algorithm. The observations marked with black circles represent the support vectors.

- K-nearest neighbors (K-nn)

  K-nn model has not typical learning process. When a new observation needs to be classified, the model identifies the k most similar observations (neighbors) from training dataset, meaning those that are closest, and assigns the observation to the class that the majority of its neighbors belong to. To select neighbors, there are various methods for calculating distances, with some of the most common being Euclidean distance and its variations

(e.g., weighted Euclidean distance or squared Euclidean distance), Mahalanobis distance [Mahalanobis, 1936], and other metrics such as L_p. Since the model relies on distance calculations, it is crucial that features are on the same scale. For this reason, the algorithm is often preceded by feature standardization or normalization.

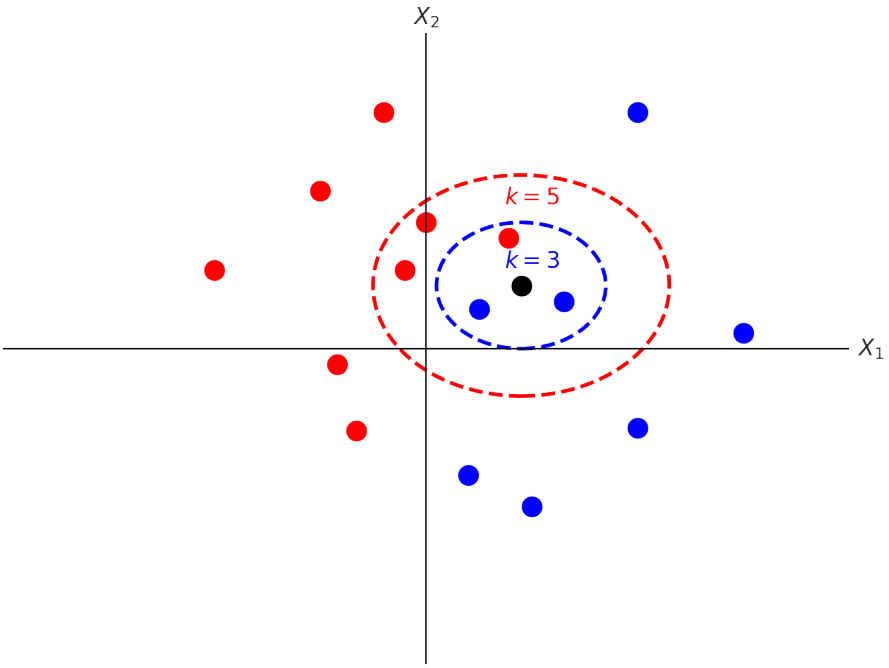

Figure 2.3: An example of applying the KNN algorithm with Euclidean distance. The black dot represents a new observation that requires classification. The red and blue dots are test set observations from two different classes.

An example of applying the k-nearest neighbors algorithm is presented in Figure 2.3. Note that with the hyperparameter k set to 3, the model will classify the new observation into the blue class. However, when the number of nearest neighbors is increased to 5, there will be more red observations, causing the new observation to be classified into the red class. This example illustrates the importance of selecting the hyperparameter and shows that in some cases, even with the available data, making a correct classification can be difficult — even for a human.

2.2.4. Ensemble learning

One approach to improving prediction quality is combining multiple models to obtain more accurate and stable results. This approach is called ensemble learning.

- Bagging

  Bagging is a technique in which multiple models are trained on subsets of data obtained

using the bootstrap method, meaning sampling with replacement. Each model is trained independently, and the final prediction in the case of classification is obtained through majority voting. It can be expressed as:

$$\hat{y} = \arg\max_{y} \sum_{m=1}^{M} \mathbb{1}(f_m(x) = y)$$

where $f_m(x)$ represents the prediction of the m-th model. Bagging helps to reduce variance and prevent overfitting, making it particularly effective for high-variance models such as decision trees.

- Random Forest

  Random forest is an ensemble learning model that use multiple classifiers to obtain better predictive performance. This model consists of numerous decision trees, each built on a random subset of features and a random subset of the data. The final prediction is made by aggregating the predictions of individual trees, typically through majority voting (see Figure 2.4) for classification tasks or averaging for regression tasks.

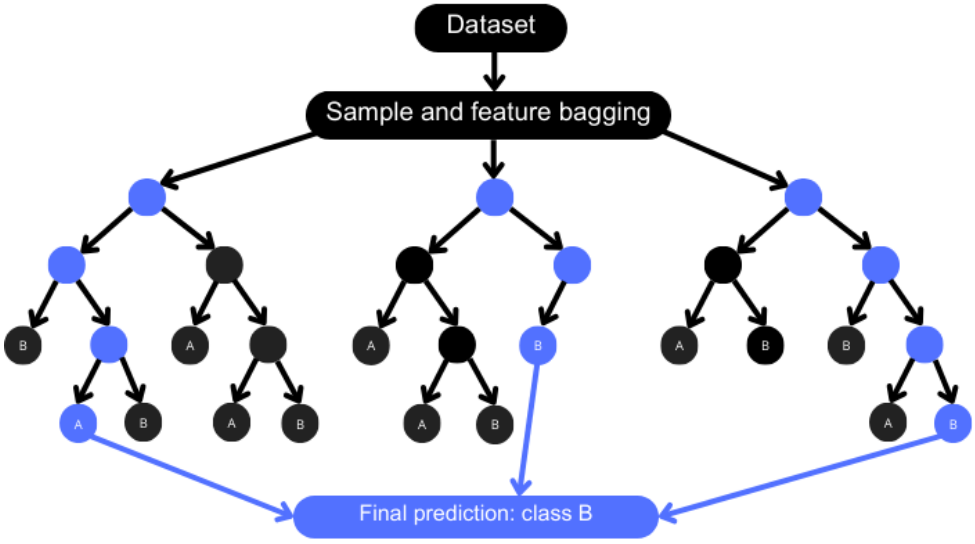

Figure 2.4: An example of how a random forest model operates with majority voting. The blue color indicates the decision paths for the same observation in individual trees that lead to the final prediction. In this case, two trees classified the observation as class B, while the third tree classified it as class A, resulting in a final prediction of class B.

- Boosting classifiers

  Boosting classifiers are a type of ensemble learning that iteratively adjusts the weights of observations based on previous errors. The method sequentially trains weak learners, giving higher weights to misclassified samples in each iteration to improve overall model

performance (see Figure 2.5). At each iteration, the weak learner is trained on a reweighted dataset, where higher weights are assigned to misclassified samples from the previous iteration. The optimization process typically minimizes an exponential or logistic loss function, as in AdaBoost or Gradient Boosting. Boosting is effective in reducing bias and variance, leading to improved generalization performance but it is also prone to overfitting.

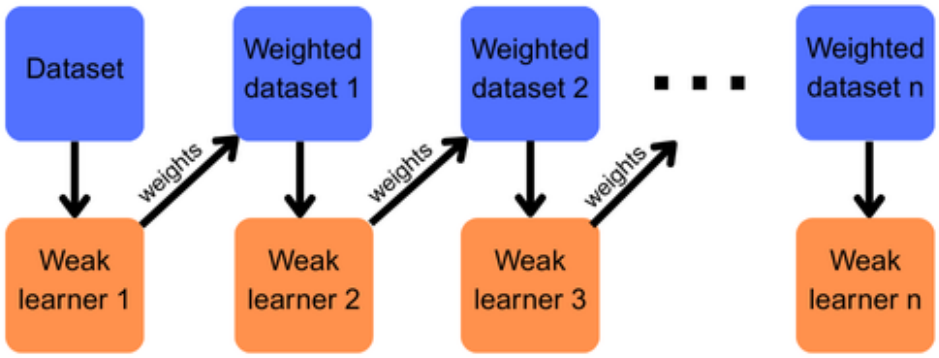

Figure 2.5: Diagram of boosting classifiers over n iterations. Each model is trained on a weighted dataset where weights are based on the errors made by the previous one. In every iteration we update weights according to the errors made by the latest model.

- Stacking

  Stacking is a complex technique that involves training multiple models on the same dataset and then using their predictions as input features for a final meta-model to obtain the final prediction. The final prediction can be expressed as:

$$\hat{y} = g(f_1(\mathbf{x}), \ldots, f_M(\mathbf{x}))$$

  where $f_m(x)$ represents the output of the m-th model, which is the probability of belonging to a given class, and g is the meta-learner that combines these predictions into a final decision. Examples of meta-learners can include logistic regression or gradient boosting.

2.2.5. Neural Machine Learning Models

One of the basic and fundamental machine learning models related to neural networks is the perceptron. The perceptron works by computing a weighted sum of the input features, which is then passed through an activation function, for example identity, ReLu or sigmoid (see Figure 2.6), to determine the output. A single perceptron was only capable of solving linear problems, which is why more complex models were developed to solve not only linear

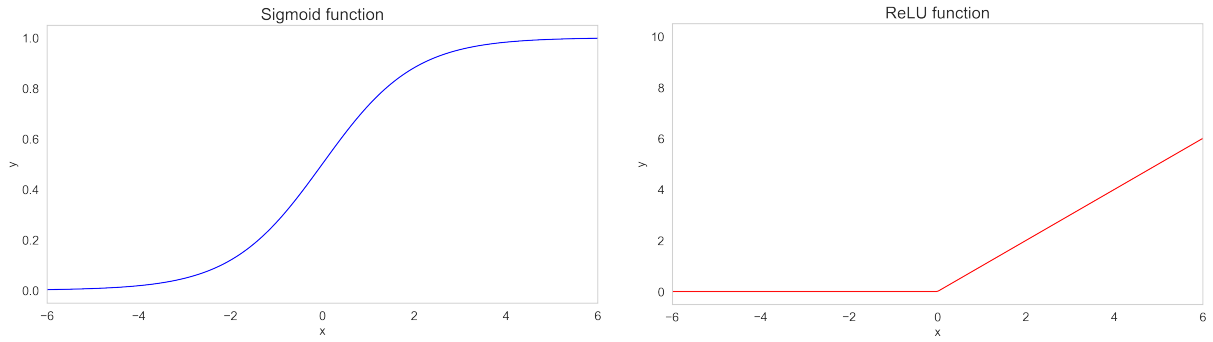

Figure 2.6: Examples of nonlinear activation functions. On the left side, the graph of the sigmoid function, on the right, the ReLU function. The sigmoid function behaves approximately linearly for arguments close to zero, whereas the ReLU function outputs zero for all arguments less than 0.

problems. Artificial neural networks (ANNs) are models inspired by the functioning of the human brain and they are more advanced perceptron. They consist of an input layer, hidden layers (0 or more), and an output layer. Each layer is composed of perceptrons, where each perceptron is connected to some perceptrons in the next layer, except for the output layer. The network diagram is presented in Figure 2.7. Each perceptron applies a nonlinear activation function. In the absence of hidden layers and with the sigmoid function as the activation function, the model reduces to logistic regression.

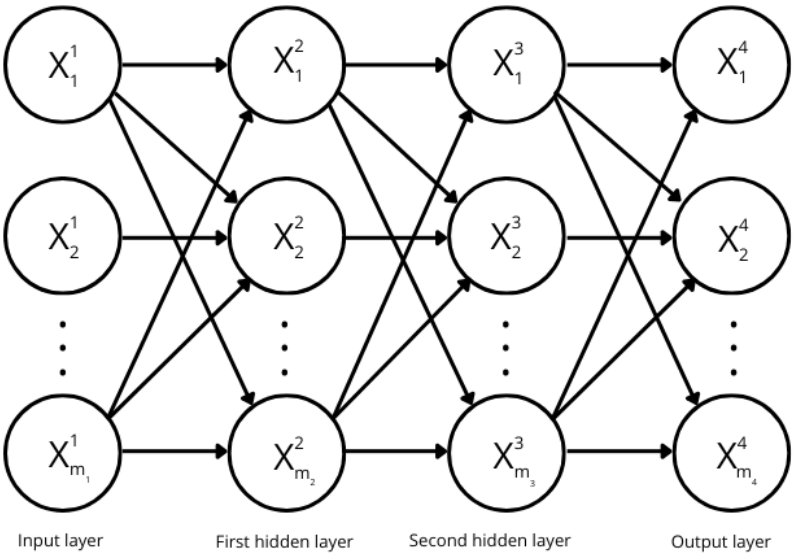

Figure 2.7: A neural network diagram with 2 hidden layers. X_i^r is the output of the i-th perceptron in the r-th layer, m_i is the number of perceptron in the i-th layer. The arrows represent connection with weights between the perceptron.

The learning process of neural network involves minimizing the loss function through backpropagation i.e. updating the weights, typically using the stochastic gradient descent algorithm [Goodfellow et al., 2016]. Additionally, various techniques are applied during training to improve model performance, such as regularization methods. One example of regularization technique is dropout, which randomly disables a fraction of neurons during training to reduce overfitting and improve the model's ability to generalize.

A wide range of algorithms and model parameters presents new challenges. The first of these is choosing the right algorithm: each one excels at certain types of problems but performs poorly on others. For example, in nonlinear tasks — especially when the number of features is large — decision trees tend to be an excellent choice, whereas linear regression fails to deliver satisfactory results. Conversely, in linear problems the situation is reversed: linear regression most often provides the best predictive quality, while decision trees can struggle to make accurate predictions. Once we have selected the appropriate algorithm, we face the next issue: model hyperparameter optimization. Hyperparameters are model settings defined before training. They control the learning process and the behavior of the model. As discussed in Section 2.2, it is precisely the parameter values that have a critical impact on the final model quality. Take decision trees, for instance—one of the fundamental parameters is the maximum tree depth. If the depth is too great, the model may overfit, if it is too shallow, the tree will fail to capture important relationships in the data. It is also worth remembering that different algorithms respond differently to parameter changes. Some, like gradient boosting, are highly sensitive to hyperparameter settings and require careful tuning, whereas others—such as logistic regression—tolerate suboptimal configurations much better and generally do not demand such precise optimization. Taking these considerations into account, it becomes clear that for any given problem one can build dozens—or even hundreds—of models by using different algorithms and hyperparameter configurations. This raises a crucial question: how can we rigorously evaluate these models and compare their performance?

2.3. Evaluating binary classification models

In this section, we will examine the most popular performance metrics for binary classifiers. We will define metrics, their applications, advantages, and disadvantages.

2.3.1. Threshold-based metrics

Definition 2.5 (Confusion matrix). Let $\mathcal{Y} = \{1, 0\}$, where we call 1 the positive class and 0 the negative class. Now, consider the following matrix known as the *confusion matrix*:

		True class	
		+	-
Predicted class	+	True positive	False positive
	-	False negative	True negative

where:

- True positive (TP) is the number of instances that were classified as positive, which were actually positive, i.e. $\sum_{i:y_i=1} \mathbf{1}(f(x_i) = y_i)$
- False positive (FP) is the number of instances that were classified as positive, which were actually negative, i.e. $\sum_{i:y_i=0} \mathbf{1}(f(x_i) \neq y_i)$
- False negative (FN) is the number of instances that were classified as negative, which were actually positive, i.e. $\sum_{i:y_i=1} \mathbf{1}(f(x_i) \neq y_i)$
- True negative (TN) is the number of instances that were classified as negative, which were actually negative, i.e. $\sum_{i:y_i=0} \mathbf{1}(f(x_i) = y_i)$

The confusion matrix summarizes the classifications made by the classifier. The rows of this matrix correspond to the classes assigned by the model, while the columns represent the actual classes. The diagonal of the matrix contains the number of observations that the classifier correctly assigned, whereas the off-diagonal elements indicate the number of instances where the classifier made a mistake. Based on the elements of the confusion matrix, other performance measures are defined.

Definition 2.6 (Recall). *Recall* is a metric that describes the percentage of observations belonging to the positive class that the model correctly classified as positive. In other words, recall describes the percentage of observations from the positive class that the model was able to detect.

$$recall = \frac{TP}{TP + FN}$$

Definition 2.7 (Precision). *Precision* describes the percentage of observations classified by the model as positive that actually belong to the positive class.

$$precision = \frac{TP}{TP + FP}$$

Definition 2.8 (Specificity). *Specificity* evaluate the performance of a classification model, defined as the percentage of observations from the negative class that the model was able to detect.

$$specificity = \frac{TN}{TN + FP}$$

Definition 2.9 (Accuracy). Accuracy measures the ratio of observations correctly classified by the model to all observations.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy provides a general indication of the model's performance, but it may not be sufficient in the cases of imbalanced datasets where one class dominates. For example, if 95 percent of the observations belong to the positive class, a model that predicts this class for every observation will have an accuracy of 95%, but in reality it will be useless. In that case precision would be prefered metric.

To extend perspective we can combine sevral metrics.

- $F_1 score = \frac{2 \cdot precision \cdot recall}{precision + recall}$ - the harmonic mean of precision and recall
- $balanced\ accuracy = \frac{recall + specificity}{2}$ - the arithmetic mean of recall and specificity (average of percentages of detected positive and negative classes)

2.3.2. Probability-based measures

Most classifiers, in addition to providing the label, also return the probability of belonging to a given class $\theta \in [0, 1]$. These models are called rankers. Examples of such models include logistic regression and naive Bayes, but decision trees or KNN can also be appropriately adapted for this purpose. In the case rankers, we need to choose a threshold, which is the value above which observations will be classified into one class and below which they will be classified into the other. Typically, the default value for this threshold is $\theta = 0.5$, but this approach may not always be the best in the case of imbalanced data. By manipulating the threshold value, we can adjust the model to suit our needs.
An example would be models that classify patients as healthy or sick. In this case, we might lower the threshold to reduce the likelihood that a sick patient will be classified as healthy. Of course, this decision will result in more healthy patients being classified as sick. The object that helps in studying these dependencies is the ROC curve.

Definition 2.10 (ROC Curve). *ROC curve* is a plot of thae recall against 1-specificity at each threshold setting θ. The Y-axis is also called the TPR (True Positive Rate), while the X-axis is called the FPR (False Positive Rate).

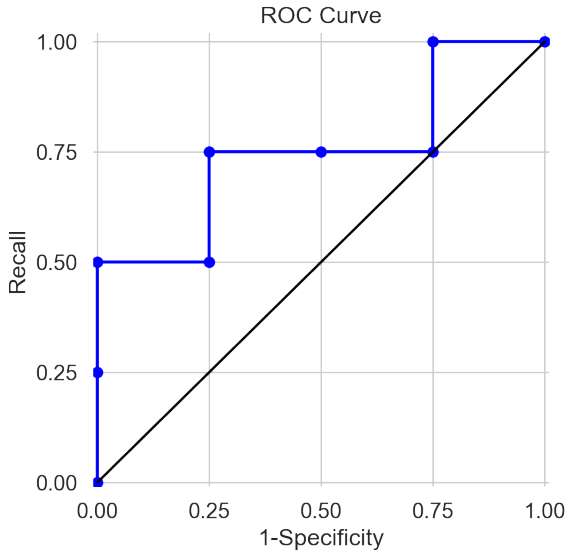

Figure 2.8: Example of a ROC curve for certain observation labels and model predictions.

Model prediction	Observation label
0.98	+
0.90	+
0.81	–
0.62	+
0.39	–
0.22	–
0.11	+
0.03	–

Table 2.1: The table shows the probability of belonging to the positive class resulting from the model for which the ROC curve is plotted and the true class label.

Figure 2.8 presents an example of an ROC curve created based on the data from Table 2.1, showing the model's predictions and the true labels for each observation. The black line in the figure represents the theoretical classification line of a random classifier. Such a model can be treated as a ranker that assigns predictions drawn from a uniform distribution over $[0, 1]$, with class assignments based purely on random chance, independent of true labels. Hence, the relationship $y = x$ holds for a random model. For more advanced models that capture meaningful patterns, this relationship should be more complex. The higher the ROC curve is above the diagonal, the better the model's performance. Based on the ROC curve, we can deduce the number of observations classified into each class as well as the number of correct classifications, which is presented in Table 2.2.

Table 2.2 presents the number of correct classifications made by the model, the recall and 1-specificity value at a given threshold, listed in the first column. This information follows from Figure 2.8 and Table 2.1. We can see that if our threshold is set, for example, between 0.62 and 0.81, then our model will classify half of the observations from the positive class as positive. If we wanted to detect all the observations from the positive class, we would need to lower the threshold below 0.11. However, this will also result in an increase in the False Positive Rate (1-specificity), meaning that more observations are incorrectly classified as positive (3, in the

Threshold	Correct classifications	Recall	1-specificity
$[0.98, 1]$	4	0%	0
$[0.9, 0.98)$	5	25%	0
$[0.81, 0.9)$	6	50%	0
$[0.62, 0.81)$	5	50%	0.25
$[0.39, 0.62)$	6	75%	0.25
$[0.22, 0.39)$	5	75%	0.5
$[0.11, 0.22)$	4	75%	0.75
$[0.03, 0.11)$	5	100%	0.75
$[0, 0.03)$	4	100%	1

Table 2.2: Table describing the number of correct classifications and recall on the established threshold.

case of a threshold between 0.11 and 0.03).

The ROC curve allows for selecting an appropriate threshold, which is crucial for tailoring the model to its applications. For instance, in the case of a model diagnosing patients, the threshold is usually lowered to reduce the probability of classifying a sick patient as healthy. A metric closely related to the ROC curve is the AUC.

Definition 2.11 (AUC). *AUC* (area under the curve) is metric summarizing model performance as the area under the ROC curve.

AUC metric ranges from 0 to 1, with a value of 1 indicating perfect classification and 0.5 corresponds to a random model.

2.4. Model Comparison

Another aspect of model evaluation is the framework used for their comparison. In Section 2.1, we discussed the necessity of splitting the dataset into training and testing subsets. This approach is commonly referred to as hold-out in the literature. However, it can sometimes be inadequate due to inherent limitations. First, a standard split means that a portion of the data—typically around 20%—is excluded from the model training process, which can significantly degrade performance when the number of observations is small. Second, the random selection of samples for the training and testing sets affects result stability—the model's performance

can vary depending on the specific split. It is for these reasons that more advanced validation methods have been developed, enabling full utilization of available data and providing more reliable estimates of model quality.

Definition 2.12 (K-fold Cross-validation). K-fold cross-validation is a technique in which the data set D is divided into k disjoint parts, called folds, i.e. $D_1, \ldots, D_k \subset D$, $D_1 \cup \cdots \cup D_k = D$ and $D_i \cap D_j = \emptyset$ for $i, j = 1, \ldots, k$ where $i \neq j$. Next, for $i = 1, \ldots, k$, we build our model using the set $D \setminus D_i$, and then evaluate the model using previously unseen data D_i.

Using the k-fold cross-validation technique, we obtain k values of the selected metric. Scheme od cross-validation is presented in Figure 2.9.

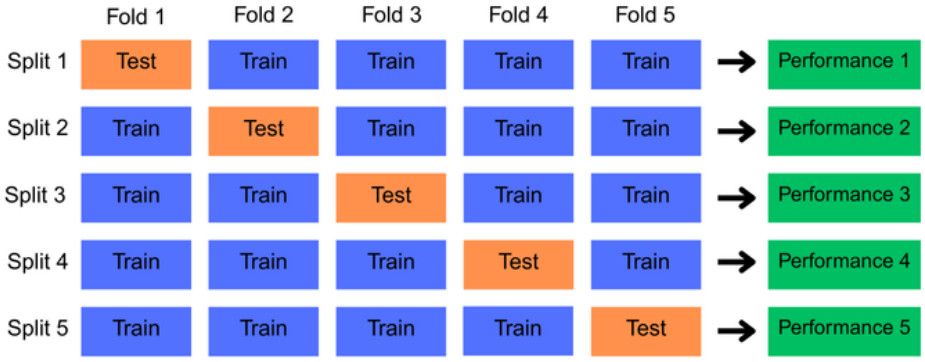

Figure 2.9: The principle of cross-validation technique using a 5-fold split.

These results are usually aggregated into a single value, for example, by using the arithmetic mean of the obtained results. However, we usually create multiple models to solve the problem. Therefore, for each model, we obtain a value (or a set of values) that we need to compare. Most of popular metrics, such as AUC, provide the ability to rank models from those that achieved the best results to those that performed the worst. However, the differences between the results of these metrics do not provide any additional interpretations or information beyond which model achieved a better score. Additionally, it often happens that the metric values are very close to each other, making it difficult to assess significant differences between them.

Another issue worth mentioning is assessing whether the differences in model outcomes are truly significant or merely arise from noise in data. Several approaches to this problem have been developed. One of them is 5x2cv t-test [Dietterich, 1998]. The procedure consists of performing 2-fold cross-validation five times. In each iteration, two models are trained on one fold and then tested on the other. Then we use paired t-test, where the standard deviation comes from the

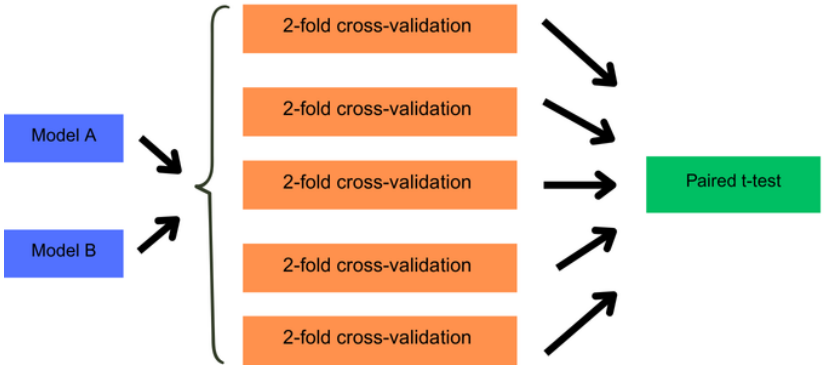

Figure 2.10: Diagram of the 5×2 CV t-test procedure. First, 2-fold cross-validation is performed independently five times, and then a paired t-test is applied.

cross-validation. The scheme of this procedure is presented in Figure 2.10.

Recognizing certain limitations of the 5x2cv t-test, further refinements have been proposed to improve the reliability and robustness of model comparisons. Notably, Alpaydin [1999] introduced the combined 5x2cv F-test, which modifies the original procedure by formulating an F-statistic that aggregates the variance information across the five iterations of 2-fold cross-validation. Unlike the t-test, which relies on averaging paired differences, the F-test captures the total variability of performance differences more effectively, providing a stronger statistical basis for hypothesis testing. This makes the method more robust, particularly in scenarios where the assumption of equal variances between models may not hold. Another significant advancement was proposed by Bouckaert [2003], who proposed a calibrated test specifically designed for comparisons based on cross-validation, where the rate of false alarms can be excessive. His method adjusts the critical values used in the test by introducing a more conservative decision threshold for assessing the significance of performance differences. This approach reduces the likelihood of mistakenly declaring two models different when their true performances are comparable, resulting in a more reliable model-selection process in practice. Despite these improvements, both the combined 5x2cv F-test and Bouckaert's calibrated procedure inherit the original design limitation of the 5x2cv framework—namely, the ability to compare only two models at a time. As a result, when multiple models need to be evaluated simultaneously, more complex statistical techniques are required.

3. Elo-based Predictive Power (EPP) - definition

In this chapter, introduce the definition of the EPP measure. We explain its formulation, how it is calculated, and its significance in assessing the performance of machine learning models, particularly in scenarios where traditional metrics may not provide a complete picture. All definitions and properties in this chapter come from Gosiewska et al. [2022].

The meta-measure EPP is based on the Elo rating system, which is commonly used in chess, soccer, and computer games. It is a ranking system designed to reflect the relative skill levels of players. After each match between players, the ranking is updated based on two key pieces of information: the players' current ratings and the match outcome. The winner gains points, with the amount depending on the strength of the opponent —defeating a higher-rated player results in a greater point gain. Similarly, the losing player forfeits points, with the amount lost being proportional to the opponent's rating. One of the advantages of such ranking systems is their probabilistic interpretation of rating differences between players, meaning the probability that one player will achieve a higher score than another. Another advantage of the Elo ranking system is that it does not require every player to compete against all others to establish a ranking. It allows for skill comparison between players who have never faced each other in a match. Some of these advantages form the foundation of the EPP score.

3.1. EPP for benchmark of models

Let $\mathcal{F} = \{f_1, \dots, f_m\}$ be a set of m models. For a fixed dataset D, we perform k-fold cross-validation using any chosen metric M. Let $\mathcal{C} = \{C_1, \dots, C_r\}$ denote the set of split of cross-validation. We obtain the values of the selected metric for each of the k rounds (cross-validation split) between every pair of models (see Figure 3.1).

Let introduce a notation for the result of the k-th round between models i and j as

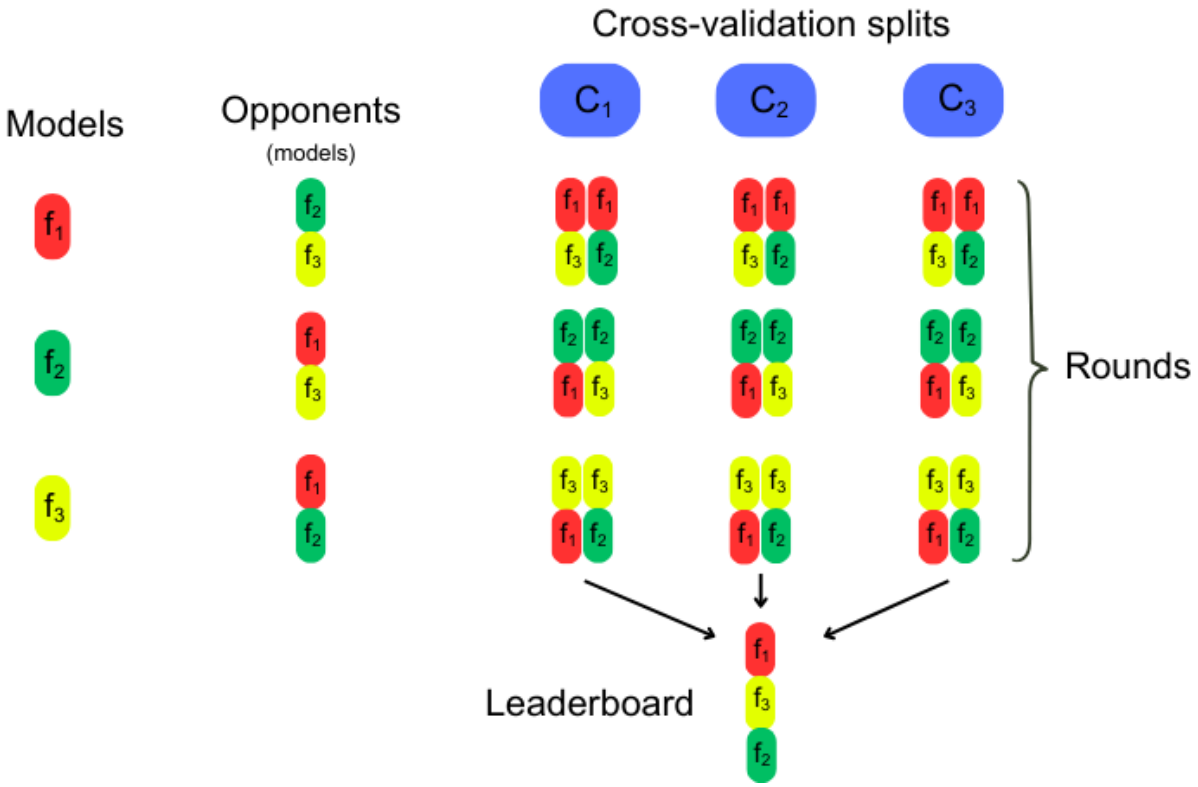

Figure 3.1: The scheme for creating the EPP ranking. On a dataset divided into 3 cross-validation folds (C_1, C_2, C_3), we compare three machine learning models — f_1, f_2, and f_3. Every rectangle represents the metric of the model in the cross-validation split.

$$y_{i,j}^k = \begin{cases} 1 & \text{where model } f_i \text{ has higher value of chosen metric than model } f_j \\ 0.5 & \text{where model } f_i \text{ has the same value of chosen metric as model } f_j \\ 0 & \text{otherwise} \end{cases}$$

We assume that a higher value of the selected metric indicates a better result. Notice that metrics usually take continuous values; therefore, in such cases, the probability of a tie is 0. Next, observe that the estimator of the probability of model f_i achieving a better result than model f_j in a random cross-validation split is expressed as

$$p_{i,j} = \frac{\sum_{k=1}^{r} y_{i,j}^k}{k}$$

.

Definition 3.1 (Odds). The odds(i,j) are odds that model f_i has a better value of chosen metric then model f_j and are expressed as

$$\text{odds(i,j)} = \frac{p_{i,j}}{1 - p_{i,j}}$$

.

Definition 3.2 (EPP). The β_{f_i} and β_{f_j} are EPP Meta-Scores for models $f_i, f_j \in \mathcal{F}$ respectively if they satisfy the following property

$$\log \frac{p_{i,j}}{1-p_{i,j}} = \beta_{f_i} - \beta_{f_j},$$

where $p_{i,j}$ can be estimated $\hat{p}_{i,j}$ in two exploratory variables logistic regression of the form

$$\log \frac{\hat{p}_{i,j}}{1-\hat{p}_{i,j}} = \hat{\beta}_{f_i} x_{f_i} + \hat{\beta}_{f_j} x_{f_j}, \quad \text{where } x_{f_i} = 1 \text{ and } x_{f_j} = -1,$$

where $\hat{\beta}_{f_i}$ and $\hat{\beta}_{f_j}$ are estimated EPP value. For simplicity, in the remainder of the work, we will refer to them as EPP values.

Definition 3.3 (Leaderboard). The EPP Leaderboard for data set D is the set of EPP values fot the set of m models $\mathcal{F} = \{f_1, \dots, f_m\}$ is

$$L_F^D = (\hat{\beta}_{f_1}, \dots, \hat{\beta}_{f_m})$$

3.2. The properties of EPP

In this section, we discuss the key EPP properties in the context of further experiments.

One of the biggest advantages of the EPP measure is the probabilistic interpretation of the differences between the performances of models. Moreover, this property follows almost directly from the definition of the measure itself. Let us return to Definition 3.2 of EPP. Notice that the left-hand side of the equation can be written as $logit(\hat{p}_{i,j})$:

$$logit(\hat{p}_{i,j}) = \log \frac{\hat{p}_{i,j}}{1-\hat{p}_{i,j}} = \hat{\beta}_{f_i} - \hat{\beta}_{f_j}. \tag{3.1}$$

After solving for the value $\hat{p}_{i,j}$ from the equation (3.1), we obtain the probability that model f_i will achieve a higher score than model f_j:

$$\hat{p}_{i,j} = \text{invlogit}(\hat{\beta}_{f_i} - \hat{\beta}_{f_j}) = \frac{\exp(\hat{\beta}_{f_i} - \hat{\beta}_{f_j})}{1 + \exp(\hat{\beta}_{f_i} - \hat{\beta}_{f_j})}. \tag{3.2}$$

The EPP results allow for the assessment of the significance of the model's performance. There is a procedure supports the evaluation of whether the differences between the performances of the given models result from noise or are indeed significantly different. Note that the values of the EPP measure are the coefficients of a logistic regression with the intercept equal to 0. From this, it follows that our definition 3.2 can be generalized to:

$$\text{logit}(\hat{p}_{i,j}) = \hat{\beta}_{f_1} x_{f_1} + \hat{\beta}_{f_2} x_{f_2} + \dots + \hat{\beta}_{f_m} x_{f_m} \quad \text{where } x_{f_i} = 1_{a=i} - 1_{a=j}. \tag{3.3}$$

$\hat{\beta}_{f_i}$ represents the estimated value of the unknown β_{f_i} coefficients from the logistic regression involving multiple exploratory variables, where x_{f_i} indicates which model is being compared. By applying logistic regression to calculate the EPP values, we are able to obtain the significance of the EPP values, which would not be possible with raw probability values. The statistical significance of the difference between EPP for two models f_i and f_j may be tested as the null hypothesis that

$$\hat{\beta}_{f_i} = \hat{\beta}_{f_j}. \tag{3.4}$$

If the results from the individual data folds in cross-validation are independent and sample size is sufficiently, this hypothesis may be tested with Wald test [Wald, 1943] or Likelihood ratio test [Neyman and Pearson, 1928]. However, even when our results are not independent, the test results are robust and can be relied upon.

Another very useful property of the EPP measure is the ability to compare models across different datasets. This allows for evaluating the performance of algorithms that may potentially address entirely different problems. Such comparisons are made by the probability of beating the "average" (reference) model in a given dataset, which is equivalent to the coefficient $\hat{\beta}_0$. Returning to Equation 3.3, we see that the value of this coefficient is 0.

In logistic regression, the intercept corresponds to the mean. Let denote it by $\hat{\beta}_{f_{avg}}$, and we observe that

$$\hat{p}_{i,avg} = \text{invlogit}(\hat{\beta}_{f_i} - \hat{\beta}_{f_{avg}}) = \frac{\exp(\hat{\beta}_{f_i} - \hat{\beta}_{f_{avg}})}{1 + \exp(\hat{\beta}_{f_i} - \hat{\beta}_{f_{avg}})} = \frac{\exp(\hat{\beta}_{f_i})}{1 + \exp(\hat{\beta}_{f_i})}. \tag{3.5}$$

Thus, to compare the performance of two models from different datasets, it is sufficient to calculate for each of them the probability of achieving a higher score than the average model in its own dataset.

4. Methodology

This chapter presents the methodology of the experiment conducted to explore the properties of the EPP measure and its applications. The experiment was based on data from a benchmark available on the OpenML platform, the analysis of which is also included in this chapter.

4.1. OpenML-CC18 Curated Classification Benchmark

The OpenML platform provides benchmarks, models along with their results, but it is primarily used as a data repository. In the followig chapters of this work, we will use terminology consistent with the nomenclature used on the OpenML platform, as outlined in Table 4.1.

Component	**Description**	**Example**
Task	Combination of a dataset and its problem	Classification in a fixed dataset
Algorithm	A general learning method or class of models	Random Forest
Flow	Model with fixed parameters and data processing (pipeline), identified by the `flow_id`	Pipeline(imputer = SimpleImputer, estimator = AdaBoostClassifier)
Run	Process of applying a flow to a task, recording the resulting predictions and evaluations	Classification using the flow with ID 8455 on task number 3

Table 4.1: Key concepts used in OpenML. The first column contains terms used later in the work, with their description and examples provided in the two subsequent columns.

First concepts introduce the notion of a task as a set of data along with the problem associated with it, which is a key element of any benchmark. Additionally, this terminology specifies the concept of an algorithm and a specific model. By algorithm, we mean an entire class of models characterized by a similar underlying principle, while a model refers to a specific instance with defined hyperparameters configuration and data processing methods. These models can come from different libraries. For example, decision trees will be referred to as an algorithm, while a

specific decision tree built with a maximum of 5 nodes and with observations containing missing values removed will be referred to as a flow. The Table 4.1 also introduces the definition of a run. A run refers to specific predictions generated for a given task by a flow, including, among other things, the model's results. These elements will be the main components of the conducted experiment.

The experiment focus on the benchmark called OpenML-CC18 Curated Classification. The analyzed benchmark includes 72 tasks. In Figure 4.1 we show histogram of the number of target classes across tasks. We can see that each of them has at least 2 predicted classes. Additionally, regarding the issue of class balancing, each class represents at least 5% of all observations (with a minimum of 20 observations).

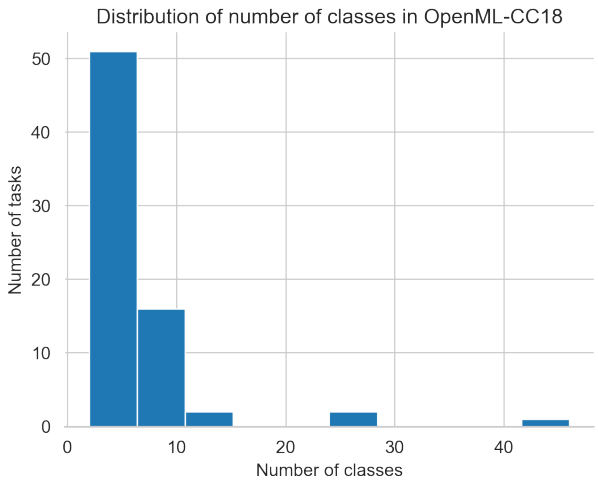

Figure 4.1: A histogram representing the distribution of the number of classes in tasks in OpenML-CC18. The X-axis represents the number of classes in target variable, and the Y-axis represents the number of tasks.

Each available dataset consists of independent observations, ranging from 500 to 100,000 in number and do not originate as subsets of larger datasets. These collections of data are not artificially generated. For example the dataset from task 14965 ('bank-marketing') comes from direct marketing campaigns of a Portuguese banking institution and was prepared by Paulo Cortez and Sérgio Moro, sourced from the UCI repository. The number of features does not exceed 5,000 (including binary-encoded categorical variables), and none of them allow for perfect prediction. The OpenML platform also provides information about the flows and their runs, i.e results from 10-fold cross-validation in the case of OpenML-CC18. In this benchmark, we have access to 3,784,673 results from 3,039 different flows.

Figure 4.2 shows that for most tasks, there are fewer than 25,000 runs available, but there are also two tasks where flows were built around 400,000 times. Based on Figure 4.3, we can observe

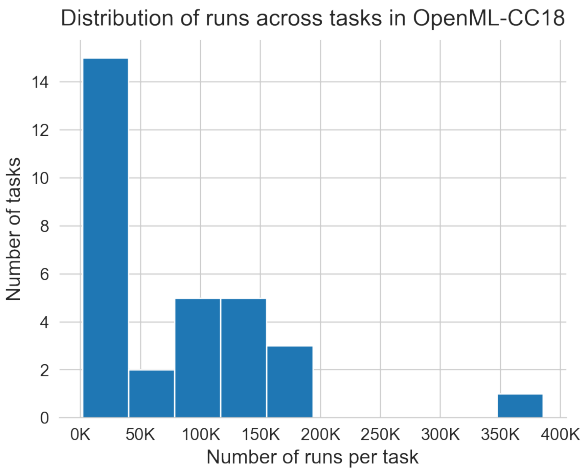

Figure 4.2: Distribution of the number of runs per task in OpenML-CC18.

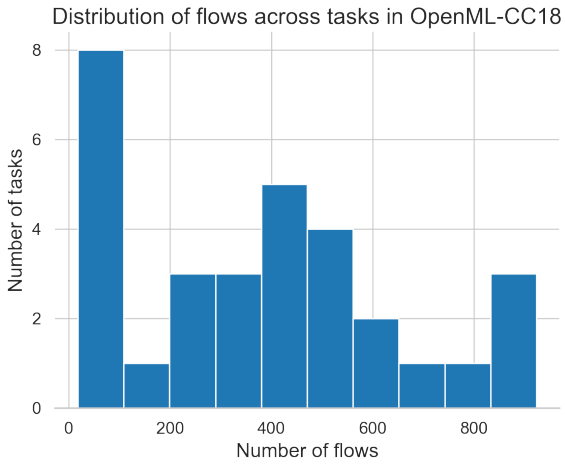

Figure 4.3: Distribution of the number of flows per task in OpenML-CC18.

that in more than a quarter of them, fewer than 100 unique flows were trained. However, some tasks have results available for over 900 flows.

4.2. Methodology of the experiment

The conducted experiment focused on tasks with a binary classification problem, which reduced the number of tasks from 72 to 35. Additionally, due to technical reasons, it was not possible to download and calculate the EPP metric for 4 tasks, so the final experiment involved 31 tasks. The tasks considered in the experiment are presented in Table 4.2, along with the names of the associated datasets and the number of observations and features.

Additionally, for clarity, the results from only one 10-fold cross-validation were considered for each flow. This way, we obtained results from 10,728 cross-validations conducted for 1,195 different flows.

Figure 4.4 shows that the vast majority of flows appear in only a small number of tasks. Nearly half of the flows occur in at most 4 tasks, and fewer than 200 flows appear in more than half of the tasks. To illustrate and deeply examine the availability of results, fill-in matrices were created.

In Figure 4.5, we can see such a matrix at the flow level. This means that each of the 1,195 columns represents a specific flow, and each row corresponds to a separate task. A cell corresponding to a given flow and task is colored yellow if a result is available for that pair, and purple if not. We can see that in this case, purple cells dominate — they account for over 76% of all entries. This indicates that many values are missing. For this reason, in some of the analyses

Task id	Dataset name	Observations	Features
3	kr-vs-kp	3196	37
15	breast-w	699	10
29	credit-approval	690	16
49	tic-tac-toe	958	10
3021	sick	3772	30
3902	pc4	1458	38
3903	pc3	1563	38
3904	jm1	10885	22
3913	kc2	522	22
3917	kc1	2109	22
3918	pc1	1109	22
7592	adult	48842	15
9910	Bioresponse	3751	1777
9946	wdbc	569	31
9952	phoneme	5405	6
9957	qsar-biodeg	1055	42
9971	ilpd	583	11
9976	madelon	2600	501
9977	nomao	34465	119
9978	ozone-level-8hr	2534	73
10093	banknote-authentication	1372	5
10101	blood-transfusion-service-center	748	5
14952	PhishingWebsites	11055	31
14954	cylinder-bands	540	40
14965	bank-marketing	45211	17
125920	dresses-sales	500	13
146819	climate-model-simulation-crashes	540	21
146820	wilt	4839	6
167120	numerai28.6	96320	22
167125	Internet-Advertisements	3279	1559
167141	churn	5000	21

Table 4.2: The table presents the tasks analyzed in the experiment. The first column contains the task ID from the OpenML platform, while the subsequent columns describe the dataset name along with the number of observations and features.

later in the study, we will aggregate results by selecting only the best-performing flow for each algorithm. In other words, for each algorithm, we will choose one flow that achieved the highest score. This approach increases the availability of results, which is illustrated in Figure 4.6.

Just like in Figure 4.5, the rows in this matrix represent tasks, but now the columns corre-

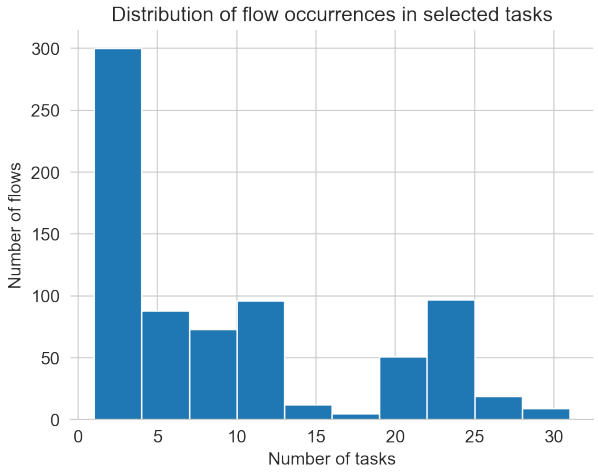

Figure 4.4: Histogram representing the distribution of the number of flows across selected tasks.

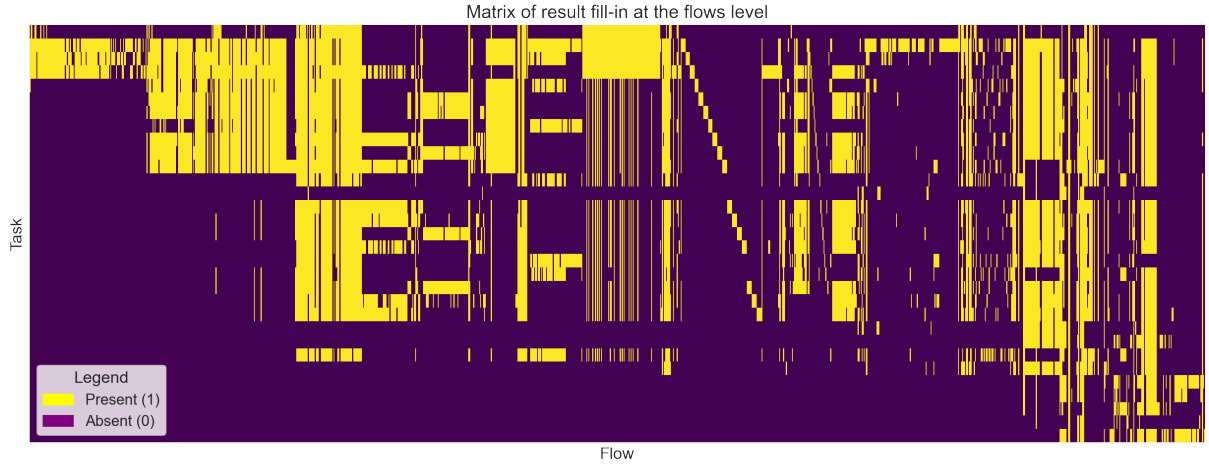

Figure 4.5: Matrix of result fill-in at the flows level. The yellow color indicates that a result is available for a given flow and task, while the purple color indicates the absence of a result.

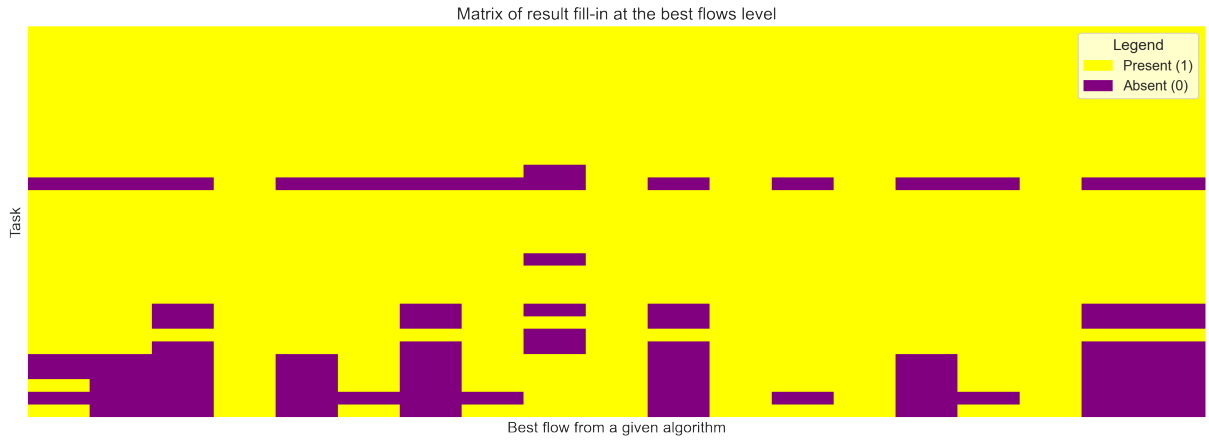

Figure 4.6: Matrix of result fill-in at the best flows level. The yellow color indicates that a result is available for a given flow and task, while the purple color indicates the absence of a result.

spond to specific algorithms. This means that if a given algorithm had at least one available flow for a task, the corresponding cell will be yellow. In Figure 4.6, we already see significantly more yellow — over 86% of the matrix — which indicates much greater result availability compared to the case shown in Figure 4.5. This will be highly useful for the analyses that follow.

4.2.1. Flows analysis

The flows used in the experiment predominantly came from four packages:

- Scikit-learn [Pedregosa et al., 2011] (16% percent of flows) is an open-source Python library (sklearn) for machine learning, offering tools for classification, regression, clustering, and more.
- weka [Frank et al., 2005] (47% percent of flows) package in Python is a wrapper for the Weka machine learning software, enabling users to apply Weka's algorithms for tasks like classification, regression, clustering, and data preprocessing directly within Python.
- Classif [Wickham, 2022] (15% percent of flows) is a package in R that provides a collection of algorithms and tools for classification tasks, offering methods for supervised learning, model evaluation, and performance assessment.
- Mlr [Probst et al., 2017] (20% percent of flows) is a package in R is a comprehensive framework for machine learning that provides a consistent interface for training, evaluating, and tuning models across a wide range of algorithms, enabling easy experimentation and model comparison.

The OpenML platform provides the names of entire flows in the form of the variable 'flow_name'. However, for a more detailed analysis and the ability to draw more complex conclusions, there arose a need to obtain algorithm names for individual flows. The OpenML platform does not directly provide such a variable, but for each flow, the name of its algorithm was embedded in the name of its flow. For this reason, the function *extract_algorithm* was created, which returns the name of the algorithm based on that full stored flow name. This function supports flows from the sklearn package and Weka, but its structure provides a relatively simple way to expand its functionality by adding new packages, introducing new algorithm classes, or modifying the assignment of a specific flow to a given class. The function takes a data frame as input, and through its parameters, we can specify:

- the libraries (packages) for which the function will extract flow names,
- the name of the column containing the flow names (default is "flow_name"),

- the name of the column that will be created to store the extracted algorithm names (default is ”model_name”),
- whether to remove rows for which the algorithm name could not be extracted.

Each of the flows is assigned one of nineteen general classes of algorithms, as shown in Table 4.3.

The principle of the function's operation is illustrated by the example in Table 4.4.

Consequently, through the application of the function, 750 flows were classified into the appropriate algorithms.

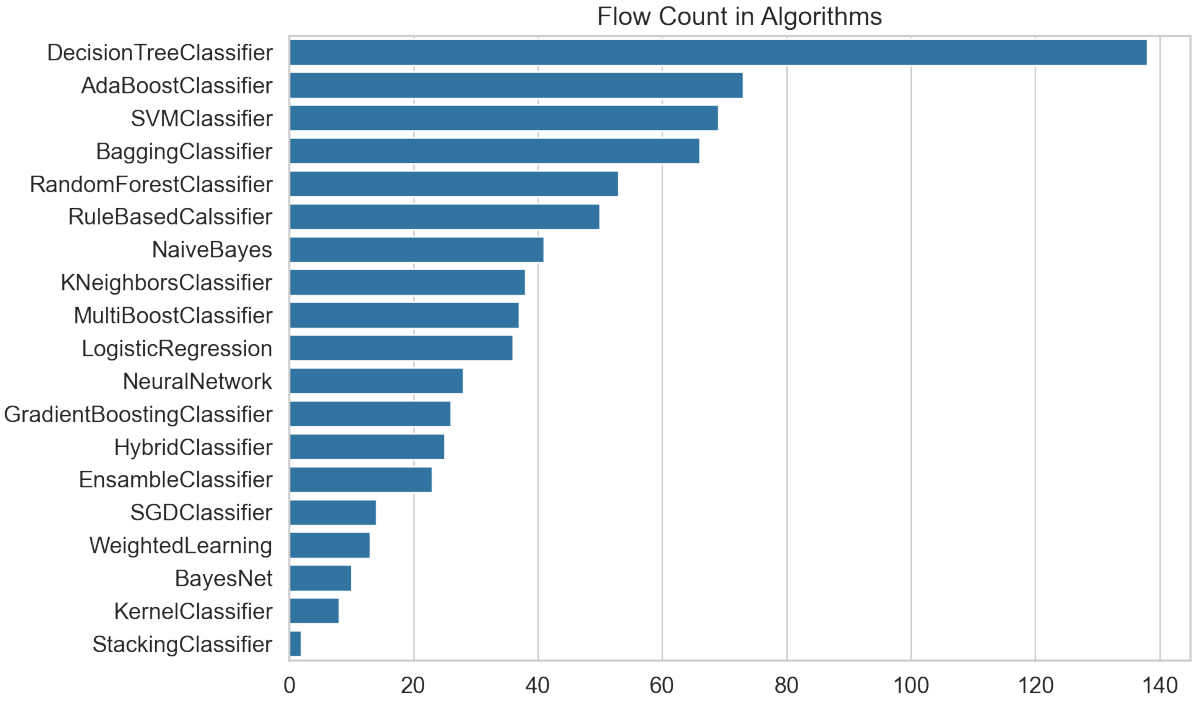

Figure 4.7: The Y-axis represents the names of the algorithms assigned by *extract_algorithm* function, and the X-axis represents the number of flows from all tasks classified into each algorithm.

From Figure 4.7, we can read the number of flows for each algorithm. We can see that the algorithm with by far the most flows is DecisionTreeClassifier – nearly 140. Additionally, algorithms such as AdaBoostClassifier, SVMClassifier, and BaggingClassifier have a considerable number of representatives. The smallest group turned out to be classifiers based on stacking. Additionally, an analysis was conducted to check the flows that appeared in the largest number of tasks, and the results are presented in Table 4.5.

It turned out that two of the flows appear in all the analyzed tasks. These are flows from the DecisionTreeClassifier and NaiveBayes algorithms with IDs 8455 and 8456.

Algortihm	**Model name extracted using the *extract_algorithm* function**
DecisionTreeClassifier	J48, REPTree, HoeffdingTree, END_ND_J48, RandomTree, DecisionStump, LADTree, BFTree, SimpleCart, Id3, A1DE, A2DE, OSDL, ExtraTree, FT, CHIRP, DecisionTreeClassifier, ExtraTreesClassifier
AdaBoostClassifier	AdaBoostM1, RealAdaBoost
BaggingClassifier	Bagging
SVMClassifier	SMO, SPegasos, LibSVM, LinearSVC, NuSVC, SVM, SVC
RuleBasedClassifier	ZeroR, JRip, OneR, PART, ConjunctiveRule, Ridor, Prism, FURIA, DecisionTable
RandomForestClassifier	RandomForest
NaiveBayes	NaiveBayesUpdateable, ComplementNaiveBayes, NaiveBayesMultinomial, HNB, GaussianNB, BernoulliNB, MultinomialNB
MultiBoostClassifier	MultiBoostAB
KNeighborsClassifier	IB, KStar, NNge
LogisticRegression	Logistic, SimpleLogistic, LBR, LR, HistGradientBoostingClassifier
NeuralNetwork	VotedPerceptron, MultilayerPerceptron, RBFNetwork, OLM, MultilayerPerceptronCS, MLPClassifier, MLPClassifier
GradientBoostingClassifier	LogitBoost, RacedIncrementalLogitBoost
HybridClassifier	LMT, NBTree, DTNB, M5P, ADTree, HyperPipes, LibLINEAR, BayesianLogisticRegression
EnsembleClassifier	RotationForest, Decorate, Vote, Grading_, VFI, Dagging, RandomCommittee, RandomSubspace, RandomSubSpace
SGDClassifier	SGD
WeightedLearning	LWL, Winnow
BayesNet	BayesNet
KernelClassifier	RBFClassifier, KernelLogisticRegression, FKC_EigenPro, FKCEigenPro
StackingClassifier	Stacking

Table 4.3: A table describing the grouping of flows to algorithms by function x. The first column contains the names of the algorithms, and the second column lists all extracted model names that have been classified under each one.

flow_id	flow_name	model_name
2010	weka.SimpleLogistic	LogisticRegression
2034	weka.Stacking_RandomTree	StackingClassifier
2058	weka.Bagging_REPTree	BaggingClassifier
2070	weka.RandomSubSpace_REPTree	EnsambleClassifier
2072	weka.CVParameterSelection_ZeroR	RuleBasedCalssifier
3920	weka.AdaBoostM1_RandomTree	AdaBoostClassifier
2074	weka.LogitBoost_DecisionStump	GradientBoostingClassifier
2094	weka.AdaBoostM1_REPTree	AdaBoostClassifier

Table 4.4: The operation of the *extract_algorithm* function illustrated with flows from the Weka package. Based on the flow_name column, the function created the variable model_name.

Flow id	Count	Algorithm
8455	31	DecisionTreeClassifier
8456	31	NaiveBayes
8693	30	DecisionTreeClassifier
8330	30	SVMClassifier
8695	30	DecisionTreeClassifier

Table 4.5: The table presents the flows that appeared in at least 30 tasks. The second column represents the number of tasks in which each flow appeared, and the last column is the classification of the flow based on the operation of function *extract_algorithm* function.

5. Experiment results

In this chapter, we present the results of the experiments conducted. Additionally, we will analyze the obtained results, from which we draw conclusions to better understand the EPP measure and relationships between algorithms. Based on the recalculated results for flows provided by the OpenML platform, and in accordance with Definition 3.2, EPP values were computed for the selected flows across each of the chosen tasks. For this purpose, the *calculate_epp_leaderboard* function from the repository EPP[1] was repository used, as it provides not only the metric value but also the corresponding p-value.

5.1. Comparison of EPP with other metrics

In the experiment, EPP metric was calculated based on two measures, AUC and recall. In this section, we compare the EPP values calculated based on the most popular metrics with the values of those metrics.

5.1.1. EPP vs AUC

The first step in the data analysis is to compare the EPP values calculated from AUC with the standard AUC metric. For this purpose, scatter plots were created, where the X-axis represents EPP and the Y-axis represents mean value of AUC for each flow. A linear regression model was fitted to the prepared dataset in order to describe the linear relationship between the EPP and AUC values. Subsequently, the regression line resulting from this model was superimposed onto the graph to visually illustrate the identified relationship.

The analysis will begin with a scatter plot of these measures for task number 9971. In Figure 5.1, we can see that the points are aligned along a line. In this case, we clearly observe a linear relationship between EPP and AUC. The Figure 5.1 has linear regression line further confirms this observation by indicating an excellent fit. The visible line is expressed by the equation

[1] `https://github.com/mgrzyb99/epp`

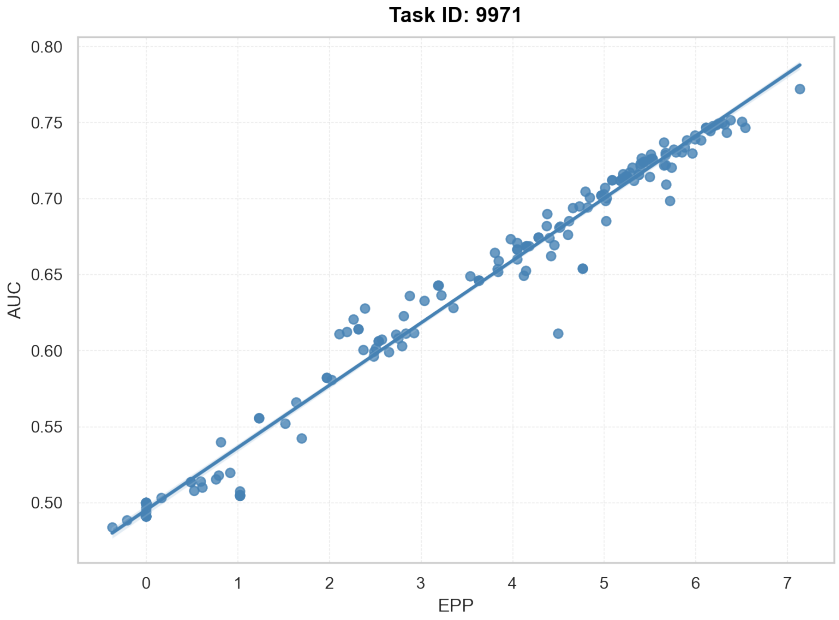

Figure 5.1: Scatter plot of AUC and EPP values for task number 9971 with a fitted line representing the result of linear regression.

$0.46 + 0.04x$ (rounded to two decimal places). This result demonstrates that the EPP metric is strongly correlated with AUC and the ranking of the flows would not change significantly if the measures were swapped. Therefore, by replacing the well-known AUC metric with EPP, we obtain very similar results while also benefiting from all the properties of EPP described above, such as its probabilistic interpretation of the differences between flow results.

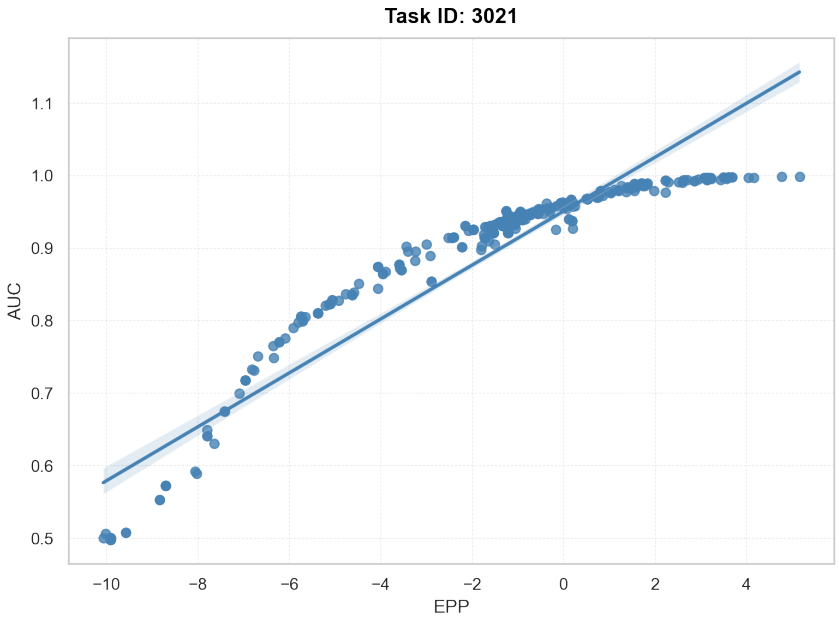

Figure 5.2: Scatter plot of AUC and EPP values for task number 3021 with a fitted line representing the result of linear regression.

In Figure 5.2 there is scatter plot connected with task number 3021, we can again observe a very strong relationship between these metrics. However, in this case, their relationship is not linear as it was previously. Instead, we see a characteristic S-shaped curve. In fact, such a relationship can be observed for most tasks. This dependence is strongly related to the definition and construction of the EPP metric. EPP is determined using logistic regression, which is based on the application of the logit function. The plot of this function (see Figure 2.6) exhibits a shape very similar to the dispersion plots shown above, which explains the observed relationship.

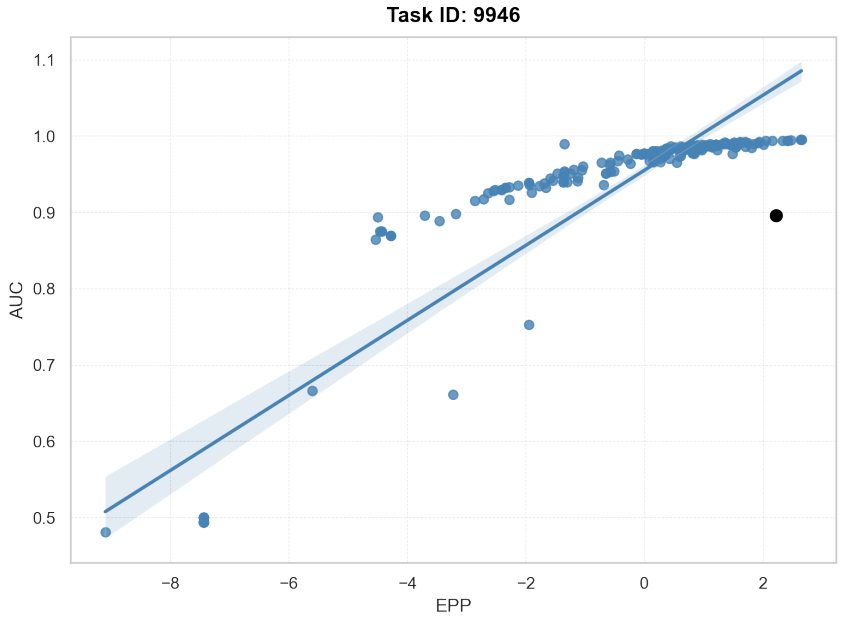

Figure 5.3: Scatter plot of AUC and EPP values for task number 9946 with a fitted line representing the result of linear regression.

Another example is task number 9946 with Figure 5.3. This case shows that a linear approximation of this relationship can be very misleading. In Figure 5.3, we see that observations with small values completely distorted the linear regression model, which poorly represents the relationship between the metrics. This plot also highlights another property of EPP. Note that the vast majority of AUC values falls within the range (0.9, 1), while most of the values are more spread out. The black-colored point marks an observation with an AUC value that stands out compared to other observations with similar EPP values. This observation corresponds to the flow with ID 2230. To better understand this phenomenon, let us examine the AUC scores across individual folds, presented in Table 5.1.

We can see that the results achieved by this flow were quite unstable, which explains its position on the plot. Although this flow won many ”duels” against others, its average AUC remained relatively low due to poor performance in some of the folds.

Fold ID	AUC value
1	0.857143
2	0.853175
3	0.904762
4	0.928571
5	0.904762
6	0.880952
7	0.863636
8	0.886364
9	0.952381
10	0.928571

Table 5.1: AUC values of flow 2230 in task 9946 per each fold.

5.1.2. EPP vs recall

To gain a deeper understanding of the relationship between EPP and other metrics, an additional analysis was conducted using a different measure. We compared EPP with another widely used metric — recall. For this purpose, we created analogous scatter plots, this time based on the newly selected metric.

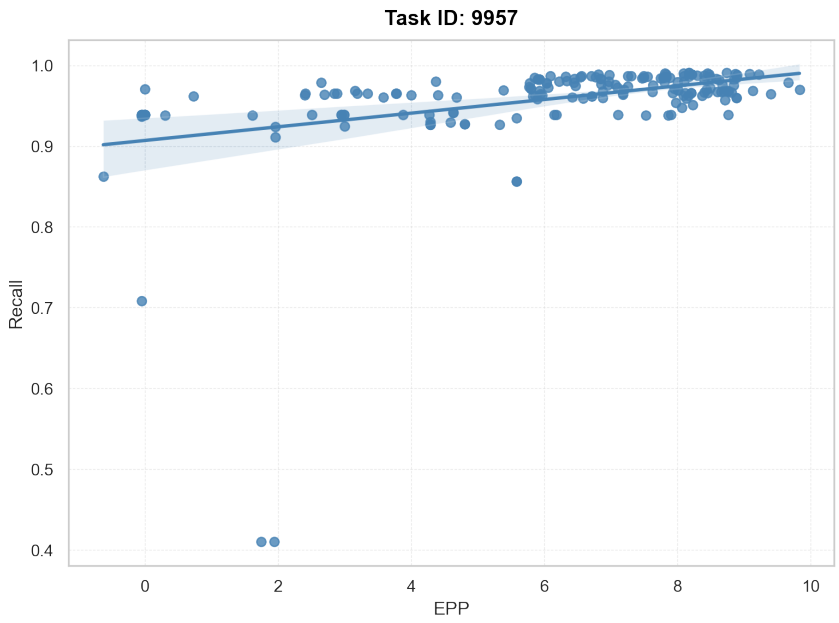

Figure 5.4: Scatter plot of recall and EPP values for task number 9957 with a fitted line representing the result of linear regression.

We will begin by comparing the EPP and recall values for task number 9957. In the case of Figure 5.4, we observe a strong linear relationship. In this case the slope coefficient is small ($a = 0.005$). The recall values here show little variation and mostly cluster around 0.95. Nevertheless, some observations deviate considerably from the fitted line, which may suggest the presence of outliers. This may be due to the instability of certain flows, as some of them may have achieved recall values with a large spread, which could have distorted the result when using the arithmetic mean for aggregation. In contrast, the EPP metric only considers which model performed better. For most of tasks, the scatter plots looked very similar. This is most likely due to the straightforward definition of the recall metric, which caused the flows to achieve very similar results.

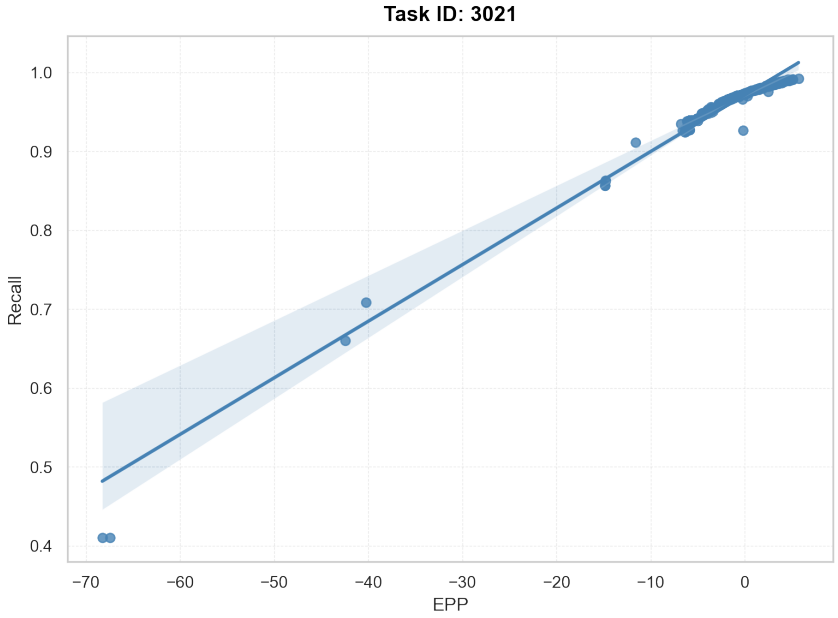

Figure 5.5: Scatter plot of recall and EPP values for task number 3021 with a fitted line representing the result of linear regression.

In Figure 5.5, we observe a very strong linear relationship with low variance. The fitted line accurately reflects the characteristics of the data. Once again, the recall values are mostly concentrated in the (0.9, 1) interval. In this case, we do not observe the characteristic ”S”-shaped curve that was present in Figure 5.2. Therefore, changing the metric in this case altered the nature of the relationship.

5.2. Analysis of the EPP measure in OpenML-CC18 Benchmark

The following part of the thesis focuses on the analysis of EPP values for the models. For this purpose, we used the EPP metric calculated from the AUC measure. The EPP values were also appropriately rescaled. Referring to the data presented in Table 4.5, we identified that there are two specific flows for which results are consistently available across all evaluated tasks. These flows correspond to two distinct classification models: flow 8455, which relates to the DecisionTreeClassifier, and flow 8456, which corresponds to the NaiveBayes classifier. The availability of these flows across the entire range of tasks presents an opportunity to establish a meaningful baseline for comparative analysis. Therefore, one of the flows – Naive Bayes – was chosen as the reference flow, meaning its EPP value was set to 0. To achieve this, we subtracted the EPP value of the aforementioned NaiveBayes flow from the EPP value of each flow within a given task. It is important to note that this transformation does not affect the probabilities of one flow achieving a better result than another. It only alters the distribution of the EPP values.

5.2.1. Probability matrices of achieving a better result

Based on the preprocessed data, we analyzed how well the algorithms perform against each other in individual tasks. To do this, within each task, the best flow from each algorithm was selected—specifically, the flow with the highest EPP value. This way, we obtained a single result for each algorithm and were able to calculate the probability of one algorithm outperforming another. Without performing this aggregation, the availability of results would be significantly lower (see Figure 4.5 and Figure 4.6). To provide a complete picture, we created full matrices that allow us to read these probabilities between any two algorithms within a given task. In these matrices the value in the cell with row number i and column number j represents the probability that the flow corresponding to row i will achieve a better result than the flow corresponding to column j. Therefore, the value in cell (i, j) is equal to 1 minus the value in cell (j, i). High values in a given row indicate strong predictive performance of the corresponding flow. Conversely, high values in a given column suggest that the associated flow is likely to be outperformed by other flows. Thus, the best flows will have high values in their rows and low values in their columns, while the weakest flows will show the opposite pattern. Since we do not compare a flow with itself, we will ignore the diagonal of the matrix, which will be marked in black.

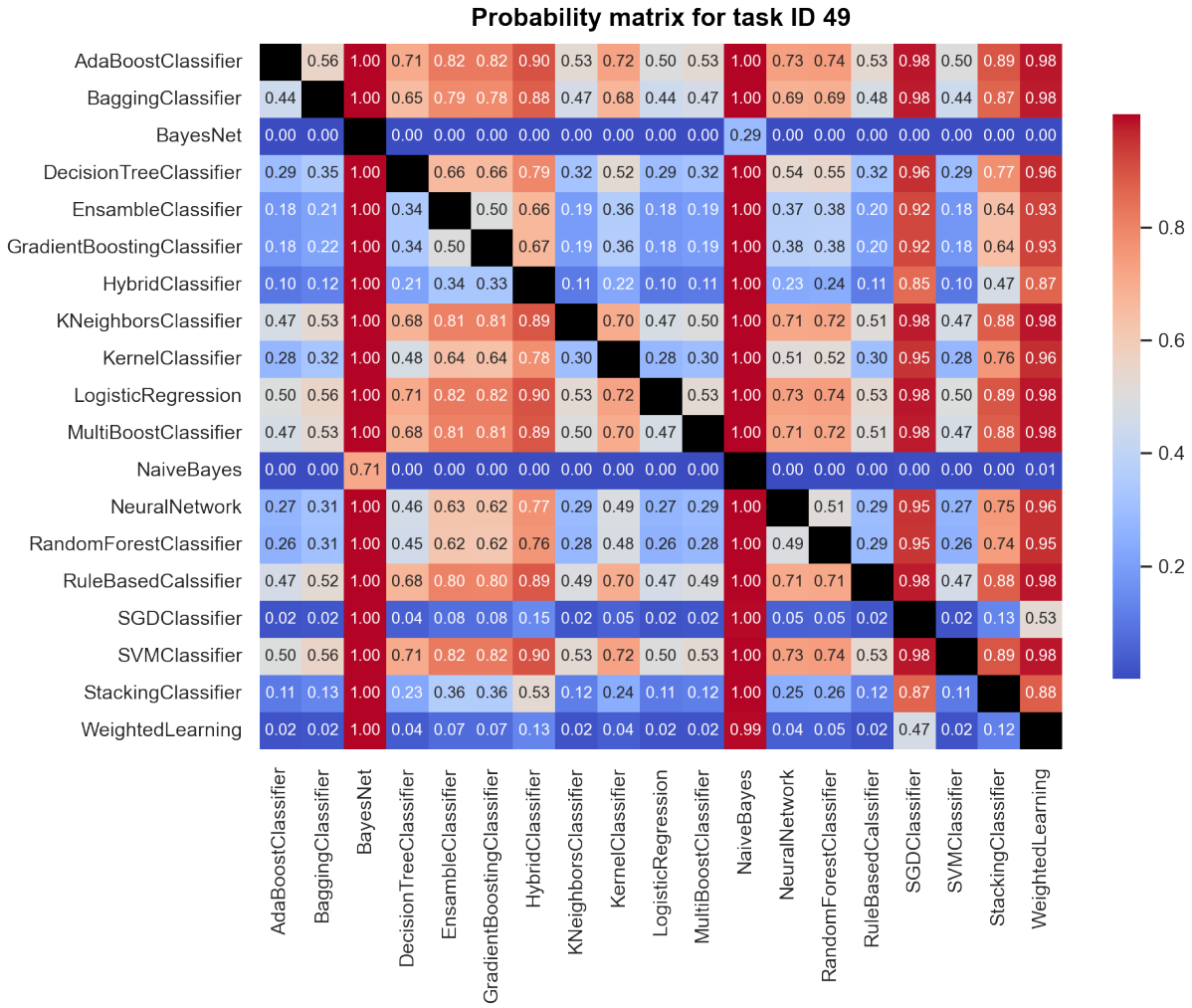

Figure 5.6: The probability matrix of achieving a better result between the best flows of a given algorithm for task 49.

In Figure 5.6, we show an example of such a matrix for the task with ID 49. In this case the AdaBoostClassifier algorithm performed the best. We can observe that all values in the first row exceed 0.5. As a result, the values in the first column are all below 0.5. Notably, in the case of two algorithms — NaiveBayes and BayesNet — AdaBoostClassifier wins with probability 1, which highlights the significant advantage of this flow for this particular task. The previously mentioned BayesNet turned out to be clearly the weakest flow, achieving a non-zero probability of winning only against the NaiveBayes flow (29%) and itself.

Let us consider another probability matrix, this time for the task with ID 7592. We observe that this matrix is characterized by many extreme values. A large portion of the flows has many values close to 1 or close to 0 in their corresponding rows. Moreover, each flow has at least one value equal to either 0 or 1. The clear best flow turned out to be a representative of the GradientBoostingClassifier algorithm, which wins against every other flow with a probability of

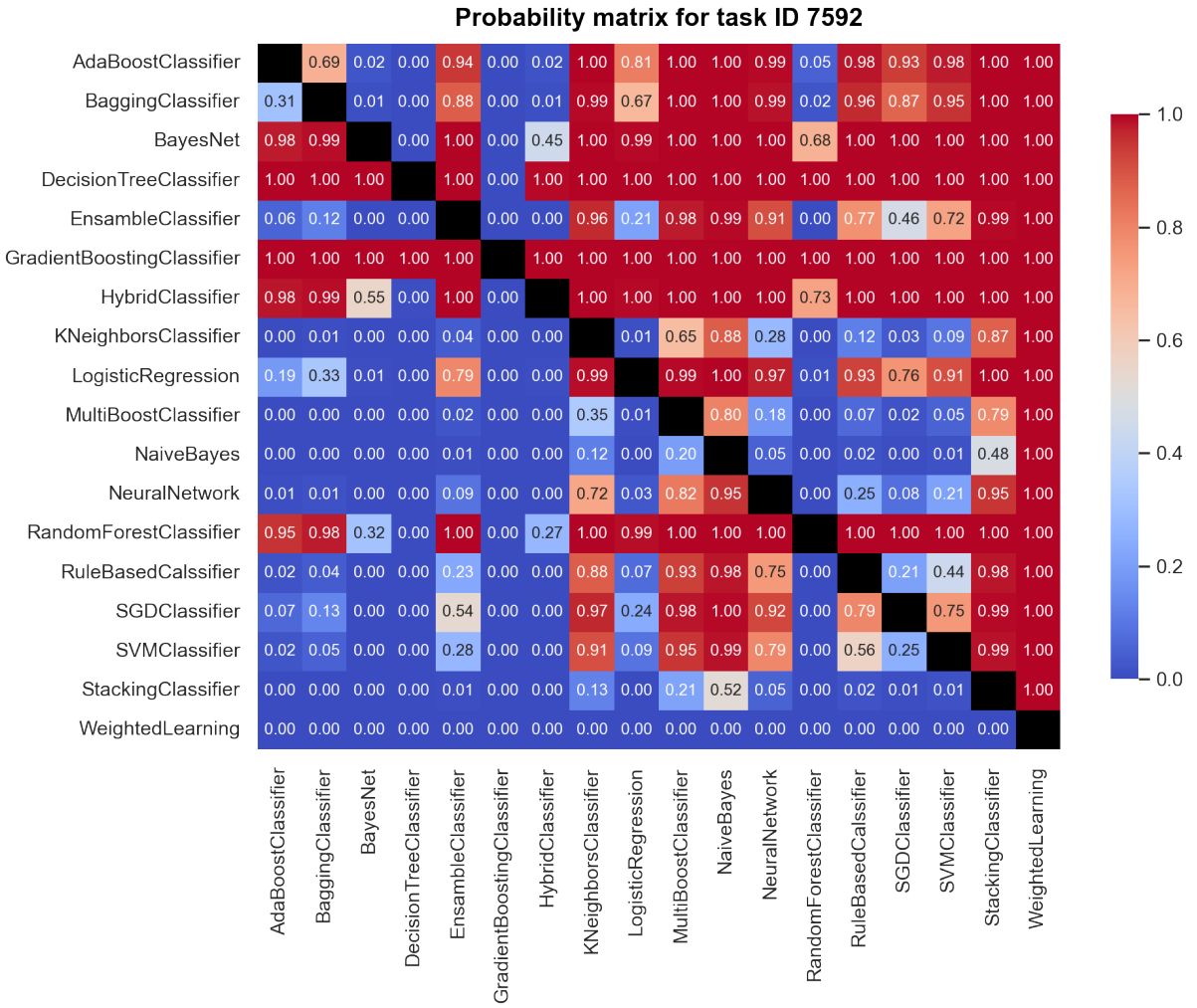

Figure 5.7: The probability matrix of achieving a better result between the best flows of a given algorithm for task 7592.

1. However, it is also worth noting that flows from algorithms such as DecisionTreeClassifier, HybridClassifier, and BayesNet also performed very well, achieving high probabilities. On the other hand, the flow from the WeightedLearning algorithm has a probability of achieving a better result equal to 0 against all other flows. Very low probabilities of defeating other flows were also observed for algorithms such as NaiveBayes and StackingClassifier.

5.2.2. EPP values distributions for algorithms across tasks

In order to examine the EPP values for individual algorithms, 31 boxplots were created for each task. These plots provided insight into the distribution of the metric and allowed for the comparison of the performance of flows from different classes. Let us now analyze two of them. The colors used in the Figure 5.8, Figure 5.9 and Figure 5.10 do not convey additional information. However, they are consistent across the charts, assigning the same color to a given

algorithm in each figure. This helps in the analysis of specific algorithms and is important in the final analysis of the work (Figure 5.12).

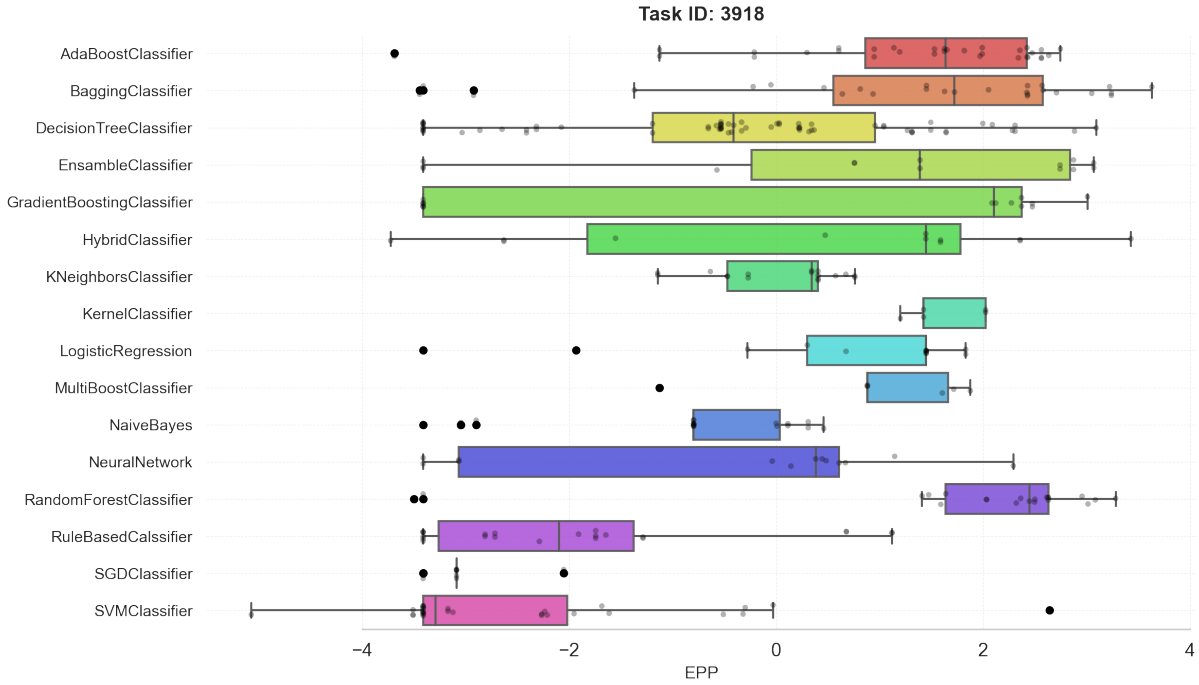

Figure 5.8: Boxplot of EPP values for each algorithm for task number 3918.

Figure 5.8 comes from the results of task number 3918. It is characterized by a small variation in EPP values. Almost all results fall within the (4,4) range. The highest median was achieved by the flows from RandomForestClassifier algorithm. Moreover, their results are relatively stable, as indicated by the narrow interquartile range. The algorithm with a slightly lower median turned out to be GradientBoostingClassifier. However, in this case, there is a very large dispersion of results. This may be due to the high sensitivity of this algorithm to flow parameters. The worst-performing flows in terms of median were those built using the SVM algorithm. The Figure 5.8 also shows several outliers, the vast majority of which represent low EPP values. This could be due to testing the flows in the early stages of development, before hyperparameter optimization.

Figure 5.9 refer to the flow results from task number 9977. In this case, we observe a different pattern of data dispersion. Most notably, the range of the data is significantly larger than in the first plot. While the vast majority of values fall within the (0, 50) interval, the inclusion of outliers results in a total range exceeding 200. This indicates that our reference flow (NaiveBayes), which achieved an EPP value of 0, performed relatively poorly. The probability of it outperforming another flow would exceed 0.5 in only 6 out of 224 remaining cases—those flows having negative EPP values. All of these cases are considered outliers, with results substantially worse than the rest. Once again, the algorithm that achieved the highest median was the

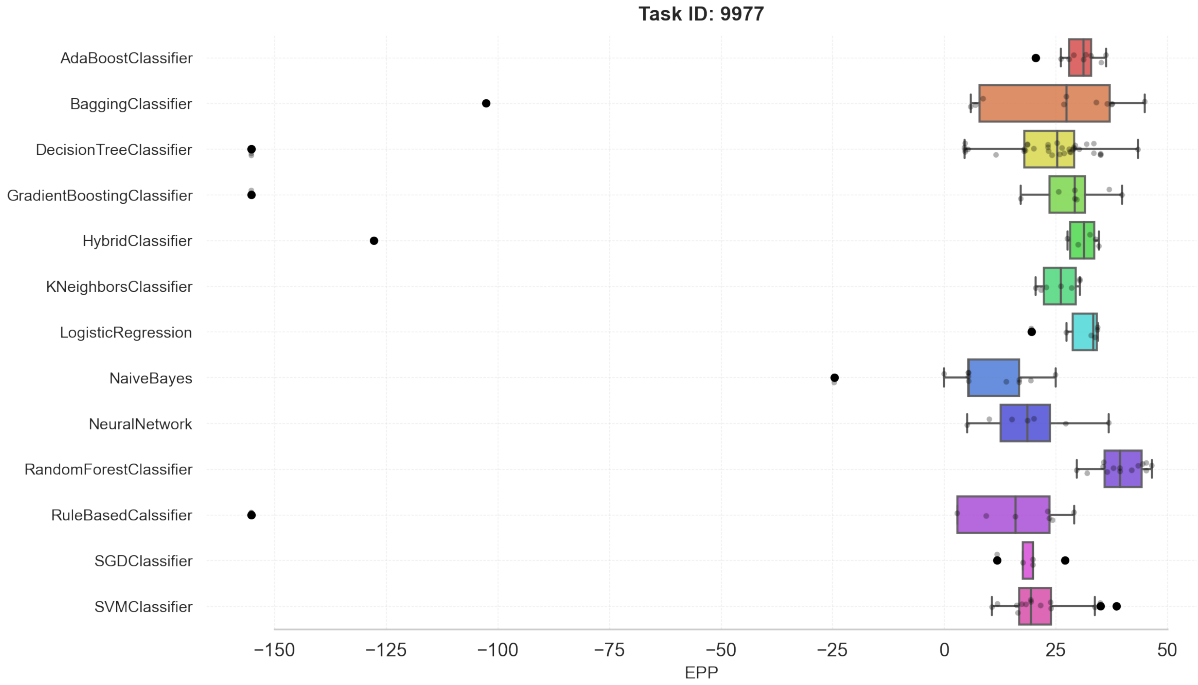

Figure 5.9: Boxplot of EPP values for each algorithm for task number 9977.

RandomForestClassifier, which also demonstrated relatively stable performance. On the other hand, the lowest medians were recorded for rule-based flows and those based on the NaiveBayes algorithm. Considering the length of the boxplot for the Gradient Boosting algorithm, we can once again infer the significant impact of hyperparameter optimization for this method.

5.2.3. Comparison of best-flow-performance distributions

The next analysis involved comparing the distribution of results for the best representatives of each flow across tasks. To do this, we once again selected one flow from each algorithm for each task, i.e., the flow with the highest EPP value. In this way, for each task, we obtained at most 19 values. Then, for each task, a box plot of these values was created.

Figure 5.10 presents the differences in EPP values for individual tasks. In some cases, an EPP value of 2 would indicate the absolutely best flow (e.g., task number 146819), while in other tasks an EPP value of 19 would represent the weakest flow. Longer boxplots for a given task suggest that algoirthm selection plays a significant role. An example of this phenomenon is the boxplot for task number 9977. Conversely, narrow boxplots, such as for task number 3913, indicate little potential for algorithm selection. We can also observe that most of the values are positive, which is due to the fact that only the best flows from each algorithm were selected. This means that these flows have a higher probability of achieving a better result than the reference flow — NaiveBayes. Some values are negative, indicating that either some algorithms had very

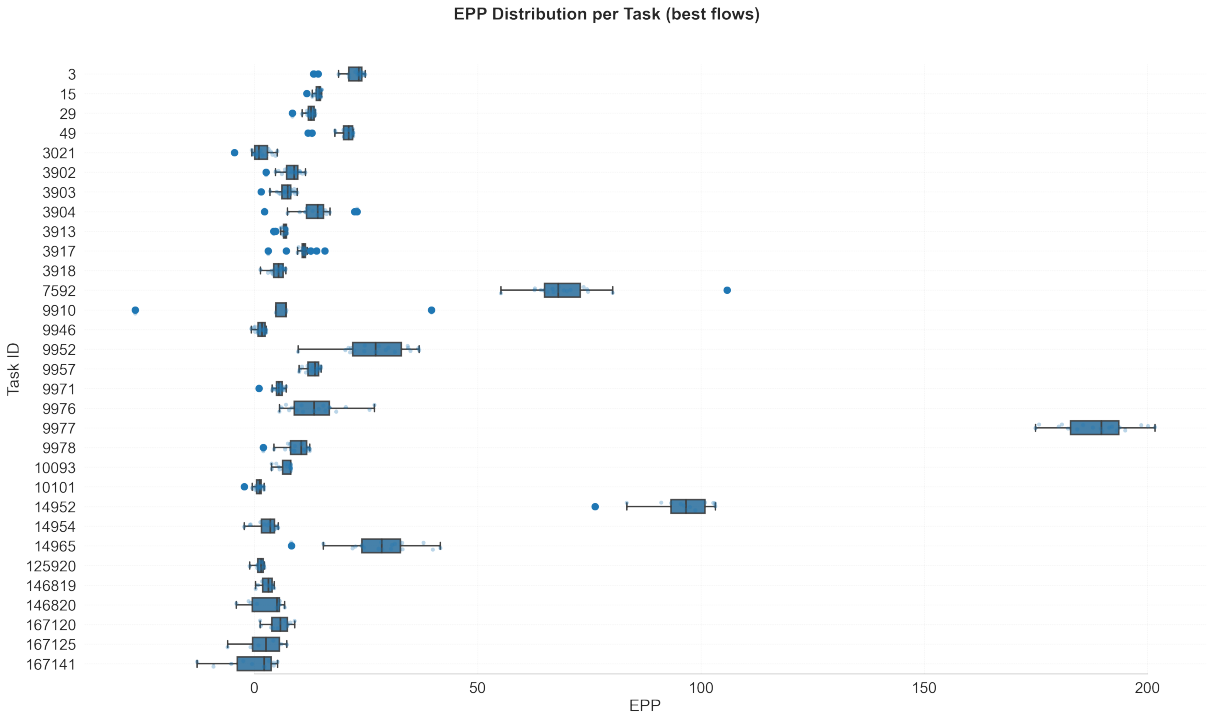

Figure 5.10: Boxplot of values for the best flows of each algorithm, divided by tasks. The X-axis represents the task number, and the Y-axis represents the EPP values.

few flows and none of them were sufficiently tuned, or that the respective algorithm was not optimal for the given solution.

5.2.4. Comparison of EPP distributions across algorithms without task segmentation

The next analysis involved comparing the distributions for individual algorithms, without division by tasks.. Figure 5.11 presents a set of boxplots, where each one represents the distribution of EPP values for flows derived from a specific algorithm, with the flow results coming from all 31 analyzed tasks.

Figure 5.11 shows that the algorithm which achieved the highest median is the RandomForestClassifier. Additionally, flows based on this algorithm attained the maximum EPP values and most often achieved a probability above 0.5 of obtaining a better result than the reference flow. In contrast, the flows from the StackingClassifier algorithm performed the worst in terms of the achieved median. The vast majority of the values are negative, which indicates that in most cases, the model performed worse than the reference flow - NaiveBayes. Each of the boxplots is characterized by a large number of outliers, which is due to the variation in task difficulty levels. We can observe that some of these outliers form smaller clusters. Most of these clusters are formed by the results for specific tasks, as the flows from a given algorithm tend to

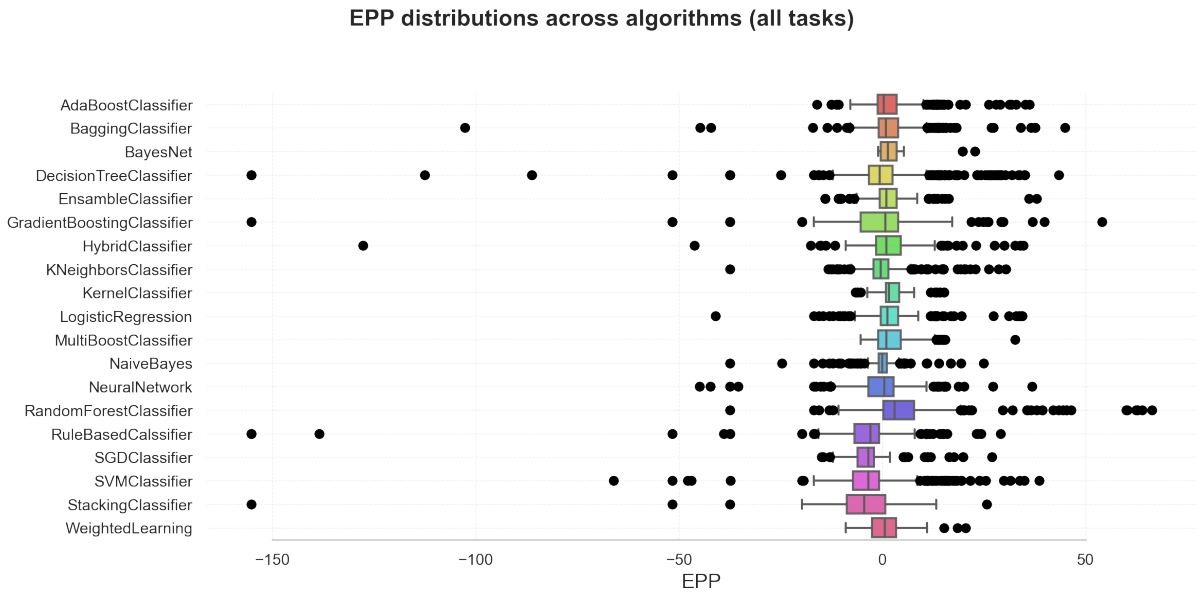

Figure 5.11: Boxplots of EPP values for a given algorithm without division by tasks. The X-axis shows the algorithm name, and the Y-axis shows the EPP values.

perform similarly on them.

5.2.5. Comparison of EPP distributions across specific flows without task segmentation

Another analysis involved focusing on specific flows without division by tasks. For this, we selected flows with results available for at least 25 tasks. Then, using boxplots, we compared their distributions, maintaining a color scheme consistent with that of the previous figures.

In Figure 5.12 it can be observed that in this experiment design, the vast majority of flows appearing in many tasks are tree-based, such as RandomForestsClassifiers, DecisionTreeClassifiers, or GradientBoostingClassifiers. This is consistent with Figure 4.7, where we can see that these flows make up a significant portion of all the flows. Additionally, it is worth noting that the results for individual flows do not differ drastically, with most of them falling within the range of (-10, 10). Analysis at the flow level confirms the excellent performance of flows from RandomForestClassifier algorithm. The highest median was achieved by the RandomForestClassifier flow with ID 8692. The weakest algorithm turned out to be LogisticRegression with ID 8673. It is also worth noting the flow from the SVMClassifier algorithm. One of them achieved one of the lowest medians among all flows and had EPP values predominantly in the negative range, while another flow from the same algorithm achieved a solid result with a median close to zero, outperforming many flows from various decision tree algorithms. This phenomenon highlights how crucial hyperparameter selection is.

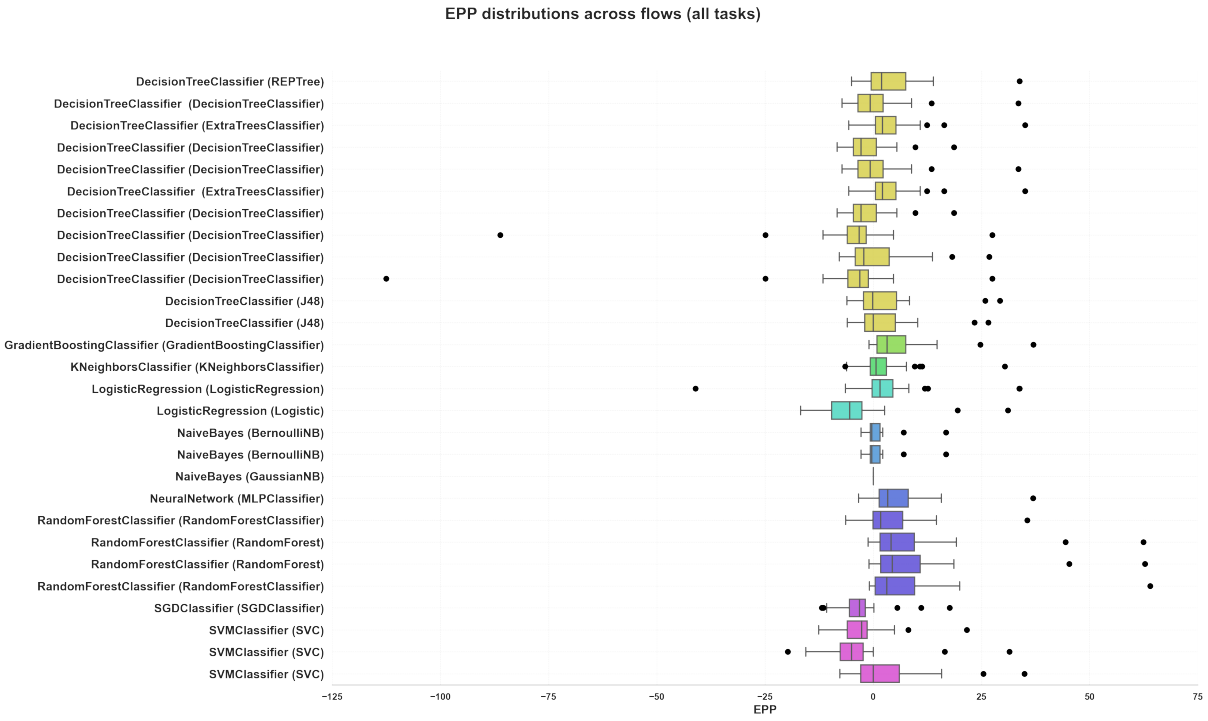

Figure 5.12: Boxplot of EPP values for flows without division by tasks. The plot is limited to flows appearing in at least 25 tasks.

6. Summary

In the main part of the thesis we conducted an in-depth comparative analysis of machine learning algorithms, extending the results presented in Gosiewska et al. [2022]. In this work, we described the fundamental concepts of machine learning. Moreover, we defined the EPP metric and presented its key advantages. The experiments were designed to compare results cross-sectionally at various levels of generality. They examined the behavior of entire classes of algorithms, individual models with specified parameters and data-processing methods, as well as the top-performing representatives of each algorithm class. In addition, the experiments evaluated model performance both on individual datasets and across multiple datasets. This approach allowed us to obtain a comprehensive view of the results and to conduct an in-depth analysis of various aspects of specific algorithms. The obtained results reveal differences in the performance of each algorithm across various datasets and underscore the importance of proper parameter selection. The most thoroughly studied models were those based on decision tree algorithms, as they appeared most frequently. Among them, the algorithm whose models typically achieved the best results was the random forest. In most experiments, models based on this algorithm consistently delivered the highest and most stable performance across a variety of tasks. Furthermore, all analyses were conducted based on data contributed by users to the OpenML platform. This approach allowed for savings in computational resources and enabled the study to be carried out on a wide variety of models originating from different packages and created by different users.

Moreover, we demonstrated the practical application of the EPP metric by comparing it against other popular model-evaluation measures. This comparison provided a deeper understanding of EPP's mechanism and highlighted its advantages over standard metrics. Finally, we developed source code and functions that enable the replication of our experiments and support further extension of this research. Potential next steps in comparing algorithms and investigating the EPP metric could include extending the study to additional software packages by enhancing the implemented function.

Bibliography

Alicja Gosiewska, Katarzyna Woźnica, and Przemysław Biecek. Interpretable meta-score for model performance. *Nature Machine Intelligence*, 4(9):792–800, September 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00531-2. URL `http://dx.doi.org/10.1038/s42256-022-00531-2`.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2 edition, 2009. ISBN 978-0-387-84857-0. doi: 10.1007/978-0-387-84858-7. URL `https://doi.org/10.1007/978-0-387-84858-7`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. ISBN 978-0-262-03561-3. URL `https://www.deeplearningbook.org/`.

P. C. Mahalanobis. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):49–55, 1936.

Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998. doi: 10.1162/089976698300017197.

Ethem Alpaydin. Combined 5×2 CV f test for comparing supervised classification learning algorithms. *Neural Computation*, 11(8):1885–1892, 1999. doi: 10.1162/089976699300016007.

Remco R. Bouckaert. Choosing between two learning algorithms based on calibrated tests. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 51–58. AAAI Press, 2003. ISBN 1-57735-189-4.

Abraham Wald. Tests of statistical hypotheses concerning several parameters when the number of observations is large. *Transactions of the American Mathematical Society*, 54(3):426–482, 1943. doi: 10.1090/S0002-9947-1943-0012401-3.

Jerzy Neyman and Egon S. Pearson. On the use and interpretation of certain test criteria for

purposes of statistical inference. *Biometrika*, 20A(1-2):175–240, 1928. doi: 10.1093/biomet/20A.1-2.175.

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.

E. Frank, M. A. Hall, G. Holmes, R. Kirkby, B. Pfahringer, and I. H. Witten. *Weka: A machine learning workbench for data mining.*, pages 1305–1314. Springer, Berlin, 2005.

Hadley Wickham. *classifly: Explore Classification Models in High Dimensions*, 2022. R package version 0.4.1.

Philipp Probst, Quay Au, Giuseppe Casalicchio, Christoph Stachl, and Bernd Bischl. Multilabel classification with r package mlr. *The R Journal*, 9(1):352–369, 2017. doi: 10.32614/RJ-2017-012.

List of symbols and abbreviations

SVM	Support vector machine
K-nn	K-nearest neighbors
ROC	Receiver Operating Characteristic Curve
EPP	Elo-based Predicitve Power
OpenML-CC18	OpenML-CC18 Curated Classification Benchmark

List of Figures

List of Tables