Summary

The Case for Learned Index is an exciting new way of thinking about data structures. Replacing traditional data structures with machine learning models can result in massive improvements for memory usage and look-up time. In our prethesis, we investigated this new way of thinking. We discovered that although we gained improvements in memory usage and look-up time, we also struggled with choosing the correct machine learning model.

This problem was further investigated in our pre-thesis, where we tried to determine the complexity of a dataset, and thereby choose the best fitting model for this dataset. We used a measure called Complexity Curve, which attempts to capture the complexity of the data in a single measure. The tests with these varied in results.

We further investigate this problem in this paper, where we employ meta-learning. Meta-learning aims to relate the characteristics of a problem to algorithm performance. The characteristics of our problem are several meta-features, which tries to capture different aspects of datasets. The meta-features were divided into groups of either, linearity, smoothness, topology and normality. We ended up choosing 8 meta-features to summarize the complexity of a dataset. The choice of meta-features was then evaluated using correlation of these meta-features and a Principal Component Analysis. The correlation of the meta-features tells us, whether any of the features are describing the same aspect of complexity. The Principal Component Analysis is a way to plot all the datasets into a 2-dimensional graph using all 8 meta-features. This allows us to see, how each meta-feature affects the principal components and is also a way to see if we cover different aspects of the datasets.

In order to be able to choose the correct machine learning model, several machine learning models were also investigated. K-Nearest Neighbour, Isotonic Regression, Spline Interpolation, Linear Regression, Multi-variate Regression and Neural Networks were considered. For each machine learning model, their strengths and shortcomings were identified. Each models performance on the datasets were then used to rank which model was the best fitting for a given dataset. The ranking was done through Multi-Criteria Decision Analysis using the Weighted Sum Method. This is because each model had to be evaluated on size, accuracy and inference time. Lastly, the Weighted Sum Method allows the user to also have some saying in the ranking. This is done through a weighted system for each metric, in which the user then can specify which metric to be the most important for their given problem.

A meta-learner was then constructed by using a neural network as a classifier. The goal of the meta-learner is to choose the correct model type for a given dataset based on the meta-features from this dataset. The meta-learner was constructed as a multi-class classifier.

Experiments were conducted. First, we evaluated the meta-learner using categorical-accuracy, recall, precision and F1-score. S, the meta-learners capability to work in the setting of the learned index were tested. Our results show an improvement in performance on all metrics when choosing the adaptive learned index.

Adaptive Learned Index

Jesper Hedegaard, Aalborg University Department of Computer Science
jhedeg14@student.aau.dk
Mark Holst, Aalborg University Department of Computer Science
mholst14@student.aau.dk

Abstract—The Case for Learned Index [11] proposes to replace data structures with machine learning models. The reasoning behind this is that most data structures are general purpose, whereas a machine learning model can be specialized to a specefic dataset. We propose to further specialize this idea by utilizing meta-learning. By looking at data characteristics called meta-features, we determined the complexity of datasets. Several machine learning models were tested and ranked based on their performance using Multi-Criteria Decision Analysis. A meta-learner was constructed which, based on the meta-features and the ranking of the machine learning models, can predict which model to use for a given dataset. Furthermore, we introduced the notion that different applications require machine learning models that excels at different aspects. Therefore, the user is able to specify which aspects their machine learning model should excel at. Our results showed superior performance compared to the base learned index model presented by Kraska et al. [11].

Index Terms—Data Structures, Learned Index, Supervised Learning, Data Complexity, Meta Learning

1 Introduction

Efficient data access requires optimized index structures and given different access patterns various structures can alleviate this problem. Furthermore, the dimensionality of the data can require specific structures. This paper focuses on one-dimensional data. For range queries the best method is a B-tree, and for exact queries a hash map provides the best results [11]. A new idea has emerged in indexing presented in the paper **The Case for Learned Index** by Kraska et al [11]. The idea is that index structures used today are general purpose where a more specialized structure might increase performance. They introduce the idea that index structures can be seen as models where the input is a key and the output will be the index. They introduce the notion that the structures could then be replaced by machine learning models which then is specialized for each dataset. The premise of this method is that data must be stored in a sorted data structure. Having a model that predicts the position, given a key inside a sorted data structure, approximate the cumulative distribution function (CDF). This allows retrieving the position of a key with the function shown in Equation 1 where N is the cardinality of the dataset and the function $f(key)$ is a trained machine learning model that approximates the CDF:

$$pos = f(key) * N \tag{1}$$

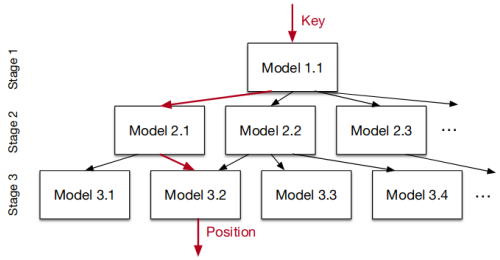


Fig. 2: Recursive Model Index [11]

The strength of this method lies in the Recursive Model Index (RMI) which allows staging of models as seen in Figure 2.
The RMI is built by having the previous model predict what data should be used for the following models in the next stage, thereby changing the dataset by assigning subsets to these models.
By each stage the models become more specialized and will reduce the last mile prediction. The different distributions contain different characteristics which make them sensitive to which models can accurately predict the underlying CDF [11]. In our pre-thesis [10] we discovered that RMI relies on a static distribution of the data to the models of each layer. Choosing the correct model can be difficult without analyzing the characteristics of the data as well as not having extensive background knowledge about machine learning models. In this paper, we propose an additional solution to this problem called **The Adaptive Learned Index**.
The system is composed of two smaller systems. An *acquisition system* and an *advisory system* shown in Figure 1. The acquisition mode is composed of a subsystem which analyses the overall complexity of a dataset given the data characteristics. Another subsystem evaluates different machine learning models

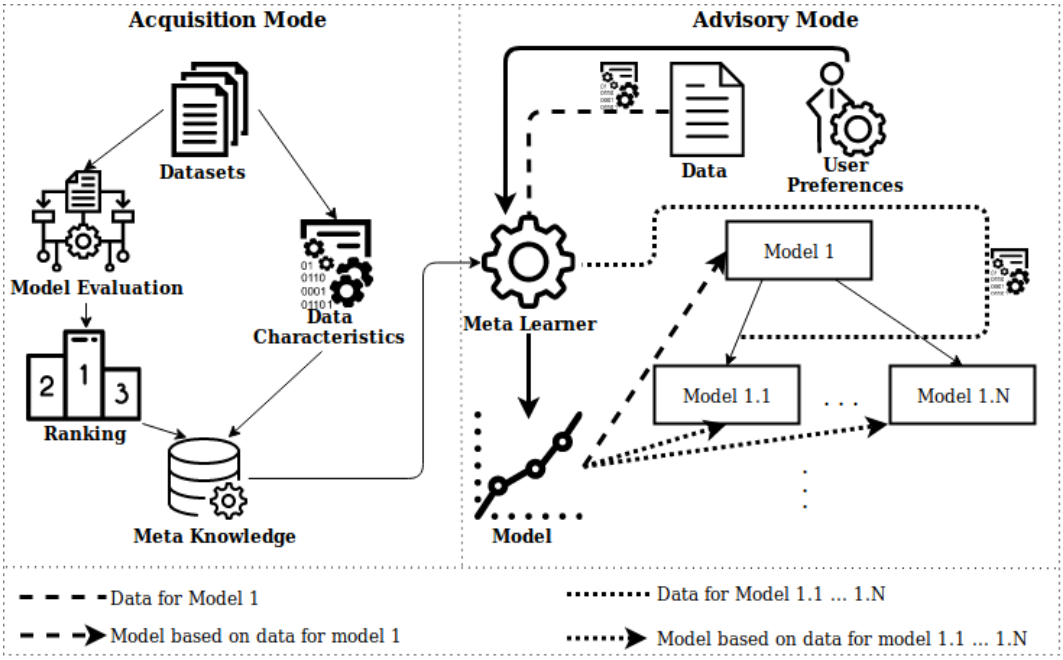


Fig. 1: Adaptive Learned Index

and compiles a ranking of the models based given datasets. This data is then fed to a meta-learner which will learn these traits. The advisory mode will then use the meta-learner to advise on which model to use. This is done by feeding the meta-learner with the data characteristics of a dataset along with user preferences on the desired ratio of speed, size and accuracy of the machine learning model. We use the advisory mode to construct the RMI by predicting the top stage model from the characteristics of the dataset.
The model is then used to distribute the data to the next layer of the RMI. The subset of data then undergoes the same process until the entire RMI is constructed.

The main contributions of this paper are as follows:

- We propose a set of meta-features (data characteristics) that will capture the overall complexity of one-dimensional datasets.
- We evaluate a set of machine learning models and identify their strengths and shortcomings when trained on different datasets.
- A meta-learner is constructed to identify the best performing model based on the data characteristics and user preferences.
- We evaluate the solution on both synthetic and real datasets showing an improvement in performance compared to the static base model presented in The Case for Learned Index. [11]

The following sections will describe the different aspects of The Adaptive Learned Index. Section 3 will describe the context of meta-learning. In Section 4 the meta-features will be selected and evaluated. Section 5 will dive into the construction of the meta-learner. The different machine learning models are introduced in Section 6. The ranking method for multi-criteria ranking is shown in Section 7. Lastly, we evaluate the system in Section 8.

2 Related Work

The Case for Learned Index [11] builds upon research on machine learning and data structures. The idea of learning characteristics of datasets and determining complexity is based on research of meta-learning. The following section will highlight the related areas.

Machine Learning: Learned index estimates CDFs of distributions. **Magdon-Ismail et al.** [13] introduces techniques for density estimation through the use of neural networks. Furthermore they introduces *"monotonicity hint penalty."* However, they claim that any sufficient machine learning model should be able to learn a CDF. Therefore, how to most effectively model a CDF is still remain an open question. Thier research is related to this paper, as we aim to aid the user in finding the best machine learning model for their distribution.
Smith et al. [18] did a study, where they systematically investigated what effect characteristics of a dataset have on machine learning models. They evaluated both neural networks and regression models. Their results showed that different models excel in different aspects.This can be related to this paper, because we investigate several meta-features from datasets. Based on these meta-features, we aim to provide an intelligent suggestion to a model for a given dataset.

Meta-Learning: Meta-learning aims to relate characteristics of a problem to the performance of

algorithms. **Bhatt et al.** [4] tries to solve the algorithm selection problem by using meta-learning. By examining the meta-learning process, they outline two phases of meta-learning; **acquisition mode** and **advisory mode**. In the acquisition mode is meta-data is extracted from the dataset. In order to correctly express a certain domain of the problem, the goal is to gather as much information as possible. Also, in the acquisition mode the evaluation of different algorithms take part. All this information is then saved in a **meta-knowledge base**. In the advisory mode, the meta-knowledge base is exploited. New meta-features are extracted from a dataset, and the meta-knowledge base is used to configure a learning system, such that a recommendation of the best available algorithm is produced. Our system builds upon this idea of meta-learning.

Meta-Features: Can be seen as characteristics of a learning problem. Generally, meta-features can be categorized into several categories. **Oreski et al.** [14] categorizes meta-features into the following categories; standard measures, sparsity measures, statistical measures, information theoretic measures, and noise measures. The meta-features are then evaluated on their contribution to a binary classification problem. **Lorena et al.** [12] further builds upon this categorization of meta-features by adding linearity- and smoothness measures. In their paper they are looking at regression problems, where linearity and smoothness can have a considerable impact on the result. Our approach utilizes both approaches as we are trying to find the best algorithm(classification) for a regression problem.

Algorithm Selection: There is a considerable amount of research on algorithm selection, of which the majority of those concerns using meta-learning, in order to find the most suitable algorithm [4], [6].
Brazdil et al. [7] made a ranking method called the **Adjusted Ratio of Ratios (ARR)**, in which the ranking of an algorithm is based on multiple measures. Specifically, it is based on the ratio of success rate and adjusted time ratio. However, as we are working on learned index, which is an alternative to traditional data structures, size is also a measure that has to be taken into account. **Tzeng et al.** [19] made a guideline for choosing the right **Multi-Criteria Analysis Decision (MCDA)** method for a given problem. They outline 3 categories of MCDA methods; **compensatory**, **partially compensatory** and **non-compensatory**. The compensatory category states that an absolute compensation between the evaluations can exist. Therefore a good performance on one criterion can counterbalance a poor performance on another. This encourages the user to determine, what evaluation measure is the most important. The partially compensatory category states that there can be accepted some kind of compensation. The problem with this category is to evaluate the degree of compensation for each evaluation. The non-compensatory category states that there exist no compensation between evaluations. Here the user decides that the inputs are enough to determine what algorithm fits best. We propose using either compensatory- or partially compensatory category, as the user might have different preferences regarding the performance of an algorithm.

Data Complexity: Our solution tries to capture the complexity of an entire dataset by using meta-features. **Zubek et al.** [24] constructed an algorithm, that attempts to decide the data complexity in a single measure. They managed to do this by constructing a **Complexity Curve** and then calculating the **Area Under Complexity Curve** value. The complexity curve is constructed by taking subsets of the entire dataset and then calculating the Hellinger Distance between the subsets and the entire dataset. As the subset grows, the complexity curve will decline. We used this measure in our pre-thesis [10] with varying results. Therefore, we look at additional meta-features in this paper. However, the complexity curve is still investigated in this paper to determine, whether or not it can give our meta-learner some valuable information.

Succinct Data Structures: Succinct data structures are about minimizing the amount of space used by the data structure, but unlike other compressed data structures, they still allow efficient query operations. Most succinct data structures focus on **H0 entropy**, which means they aim to only store the number of bits that are necessary to encode each element in the index [8]. This is related to this paper, as entropy can be seen as a characteristic. Our solution also tries to optimize data structures by using data characteristics to employ the most fitting machine learning model.

3 Meta-Learning

Meta-learning revolves around gaining knowledge about the characteristics of a given dataset. These characteristics are known as meta-features and the collection of them are regarded as meta-data. Meta-data provides information about data itself and is used for learning the learning process. We then define meta-learning as the process of making the learning process more effective and optimize it by recommending the best suited learning algorithm. Examples of meta-features could be the number of elements in a dataset, the skewness of the data distribution and linearity. There are a lot of meta-features that can be discovered from a dataset and a lot of research is focused on this area [23][15]. Most research is focused on classification problems and research on specific meta-features for regression problems is sparse and especially for one-dimensional data. We will discuss the different aspects of meta-features with regards to regression and pick meta-features accordingly.
The most important task in meta-learning is feature selection. There is no silver bullet for selecting the correct

meta-features for a given problem as per the *No Free Lunch Theorem* [17]. We thus analyze the different aspects of meta-features and select the areas that can optimize our learning problem.
Meta-features can be split into the following categories; simple, statistical and information-theoretic, landmarking, and model-based features.
Simple features revolve around the number of instances and classes in a dataset. Some algorithms are sensitive to the number of instances [22].
Statistical and information theoretic features can inform us on the distribution of the dataset along with the entropy of the attribute values [22].
Landmarking features are computed by training simple and significant machine learning algorithms that can describe properties of the dataset [3].
Model based features are computed by training decision trees without pruning them. Different properties by the induced tree are used as meta-features such as the height and width of the decision tree, number of leaves and number of nodes [3].
In The Case for Learned Index, which is a special case of regression, other features could be relevant, because the task is to completely learn the dataset (overfitting), which is not a traditional problem.

4 META-FEATURES

Selecting the best features for a given problem is difficult as there is no superset of features that explains every dataset perfectly [2]. We will investigate which characteristics identify the complexity of the dataset and find meta-features that can help measure these characteristics. The meta-features must be able to identify different aspects of the problem such that we can rely on them to select the best performing algorithm for the given dataset. We will examine whether the selected features are informative enough by analyzing their contribution to the problem.
Given the nature of the problem of one-dimensional regression, features measuring feature space, feature correlation or class separability cannot be used as meta-features. The focus will be towards the following categories; linearity, smoothness, topology, and normality.

4.1 Linearity

Measuring the linearity of a dataset can tell us whether a linear fit can be made on the dataset. If this is the case we assume the complexity of the dataset is low because a linear regression can model the data.

Correlation
The correlation coefficients of two variables measure the strength of a linear relationship between variables. A value of 0 indicates no linear relationship and a value of 1 shows a strong linear relationship. A high value is therefore assumed to indicate a simpler problem.

$$\rho_{x,y} = \frac{COV(x,y)}{\sigma x \sigma y} \tag{2}$$

Linear Residuals
This meta-feature is inspired by Lorena et al. [12]. Utilizing a fast landmarking model, in this case linear regression, we compute the mean absolute error of the residues of the model. The assumption here is that higher values equal higher complexity.

$$LR = \frac{|\varepsilon_i|}{n} \tag{3}$$

4.2 Smoothness

In regression problems, a smoother function indicates a simpler problem [12].

Smoothness of Nearest Neighbor Regressor
This meta-feature is inspired by Lorena et al. [12]. Given a dataset, we select data points that have similar output values. Given these pairs, we randomly interpolate new data points into a new dataset of size l. A nearest-neighbor regressor is then created from the old dataset. Thereafter we measure the MSE of the new data points. If the dataset is distributed smoothly the interpolated data points will conform to the same distribution. The assumption is that a high value indicates more complex data.

$$SNNR = \frac{1}{l}\sum_{i=1}^{l}(NN(x_i') - y_i')^2 \tag{4}$$

4.3 Topology

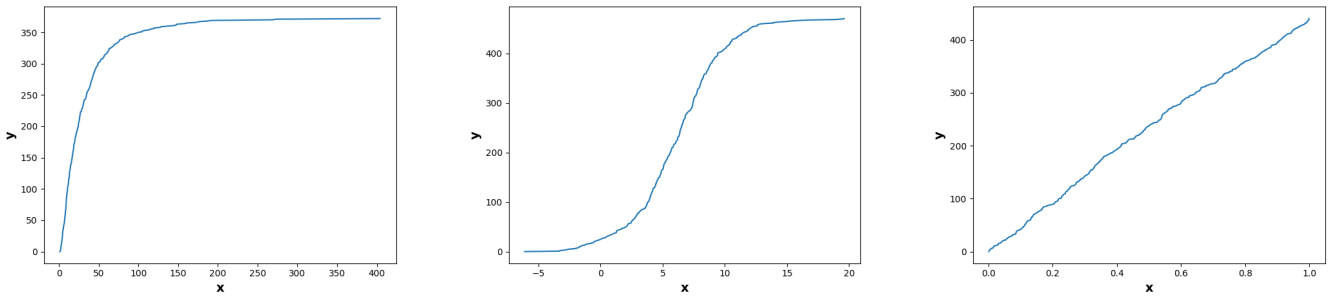
The following features will display the distribution and density of the dataset. If the data is clumped together, resulting in sparse areas, this can lead to a misleading model. These features can tell us if the distribution of a dataset calls for a complex or simple technique to model the data.

Delta Difference
The delta difference is a measure of how inconsistent the data is spread. We compute the mean delta difference ($\overline{\Delta_x}$) and calculate how many data points are considered outliers compared to the mean. A high number might suggest a more spread dataset and thus we assume that a high value equals a higher complexity.

$$DD = \sum_{i=2}^{n}(P(i)) \tag{5}$$

where

$$P(x) = \begin{cases} 1 & \text{if } (x_i - x_{i-1}) > \overline{\Delta_x} \\ 0 & \text{if } (x_i - x_{i-1}) \leq \overline{\Delta_x} \end{cases}$$



(a) Lognormal (b) Logistic (c) Uniform

Fig. 3: Example distributions

	Log-normal	Logistic	Uniform
Cor	0.8032162895102859	0.9627210988060309	0.99932690189565
LR	0.148933153740203	0.0636661509718142	0.00887126206908593
SNNR	213948080592179e-07	3.5777846313717603e-07	3.99770465422413e-07
DD	50	109	159
Var	0.0250953084139718	0.0265442105773021	0.0864490981438755
CC	0.037601355013080894	0.03397587116640183	0.02843168699553596
Kurt	6.47197677866765	0.44511455900275104	-1.22952242353569
Inst	373	471	441

TABLE 1: Meta-feature value for the datasets log-normal, logistic and uniform

Variance
Variance measures the spread of the dataset. A high spread can indicate a high complexity.

$$\sigma_x^2 = \frac{\sum_i^n (x_i - \overline{x})^2}{n} \tag{6}$$

Complexity Curve
The Complexity Curve is a combined measure of the complexity of the dataset. It constructs a probability density function of the whole dataset. Afterwards, it constructs probability density functions of subsamples of the dataset. These subsamples eventually grow in size. The probability density functions of the subsamples are then compared to the probability density function of the whole dataset using Hellinger Distance. This eventually output a function where the Area Under Curve can be used as a measure of how complex the data is. Higher value equals higher complexity [24].

$$CC(n) = H^2(P, P_S) \tag{7}$$

4.4 Normality

Some algorithms require the data points are distributed normally or that the sample size is relatively small. Normality features indicate whether we can rule out some algorithms.

Kurtosis
Kurtosis measures the combined weight of the tails of the distribution in relation to the center of the dataset. A high value means heavy tails, while low values indicate flatness. Therefore, we assume that a high value equal higher complexity.

$$Kurt = \frac{\mu_4}{\mu_2^2} \tag{8}$$

Instances
The number of examples used. This feature is used as some models are sensitive to the number of examples.

$$Inst = n \tag{9}$$

In Figure 3 examples of three different distributions are illustrated. The assumption here is, that the uniform distribution, as seen in Figure 3c, is the least complex of the distributions, while the log-normal distribution in Figure 3a is the most complex. In Table 1 all meta-features are listed with their respective values for the distributions depicted in Figure 3. Some of the meta-features in table 1 do not fit the assumptions we made above. DD and Var are notable outliers. DD should have a higher value on more complex datasets, but as seen in the table, the Log-normal distribution has the lowest value. This could however be explained by the fact that a lot of the data points in the Log-normal distribution are clustered together. In the uniform distribution the data points are more spread, which equals higher value for DD. The same reasoning can be used for the Var feature. However, meta-features such as $Corr$, CC and $Kurt$ does follow the assumptions made above.

4.5 Evaluation of Meta-Features

In order to determine whether the meta-features described above captures different aspects of the datasets, correlations between the features were calculated. The correlations can be seen in Figure 4. The darker the color, the stronger the correlation is. Blue is directly correlated while red is inversely correlated. A correlation that is worth noting is how inversely correlated the feature $linearresiduals$ (LR) are with the feature $correlation$ (Cor), as no other features have similar correlations. This however makes sense, as LR is a measure that calculate the errors of a linear regression while Cor is a measure of the strength of a linear relationship. Therefore, these will be inversely correlated. Otherwise, the plot indicates that different aspects are covered by the meta-features, as there are both direct-, inverse, and no correlations.

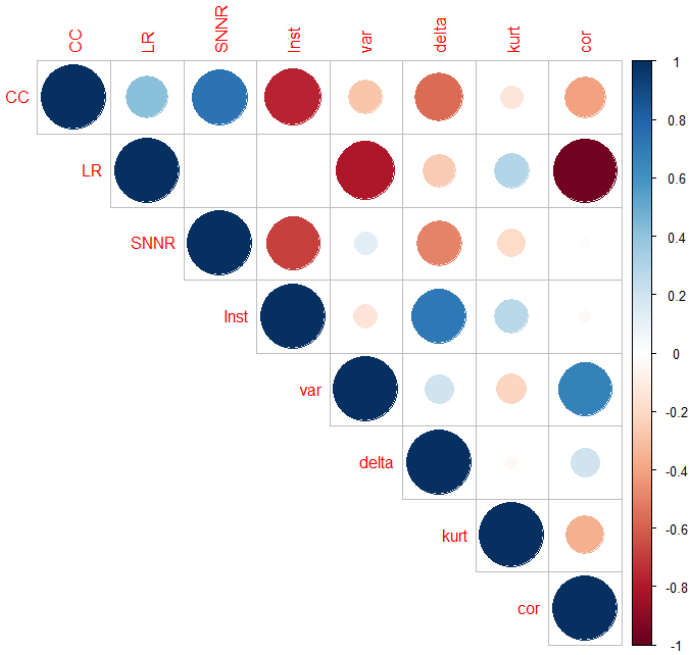


Fig. 4: Correlation between meta-features

A Principal Component Analysis (PCA) was also performed. The principal components were computed using the meta-feature values from different data distributions. Five principal components were constructed with the variance of 95%, where the two first components had $\approx 75\%$ variance combined. In Figure 5, a scatter-plot of the datasets can be seen, with the values of the two first principal components.

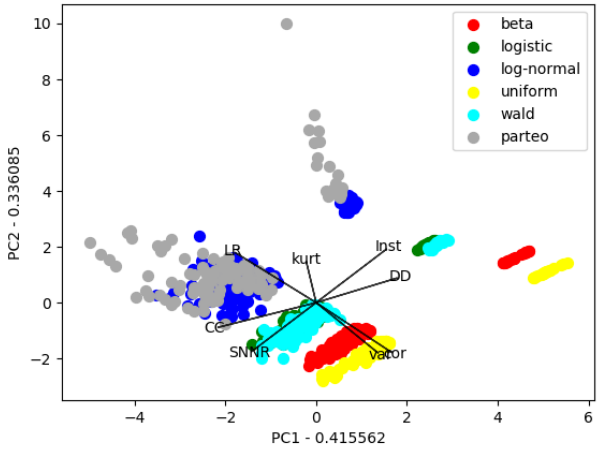


Fig. 5: Scatter plot with PCA values and all datasets

The first component is responsible for 41.6% variance, while the second component is responsible for 33.6%. The distributions are divided into four clusters. Each cluster containing up to two kinds of distributions. Based on the PCA plot, a combination of the meta-features is required in order to distinguish between the clusters of the distributions. However, the clusters seem to be most affected by LR, Var and Cor. These features focus on the complexity of the data. $Inst$, DD and $SNNR$ does not seem to contribute to the clustering but instead divides the distributions inside the clusters. These features focus on the sparsity of the data. It is worth noting that these data points represent different distribution functions and not different machine learning models. As machine learning models can be affected by the sparsity of the data, we believe that including measurements for the sparsity of datasets to be important.

In Table 2 all meta-features are listed with their impact on the first- and second principal component respectively. The features are divided into groups i.e. linearity, smoothness, topology and normality. A high value means a high direct impact on the given principal component, while a low (negative) value indicates an indirect impact on the principal component. As the value goes to 0, the less impact the given meta-feature has on the given principal component. As an example, the meta-feature CC for PC1 has a value of -0.48, which tells us that CC has a high indirect impact on this specific principal component. This is also prevalent in Figure 5. It is also clear from both Table 2 and Figure 5 that CC does not have a high impact on the PC2 because of the close to 0 value -0.19. It is worth noting that although some meta-features are grouped together, it varies how they affect the principal components. This further supports the assumption, that in order to explain the complexity of datasets, few measures are not enough to do so.

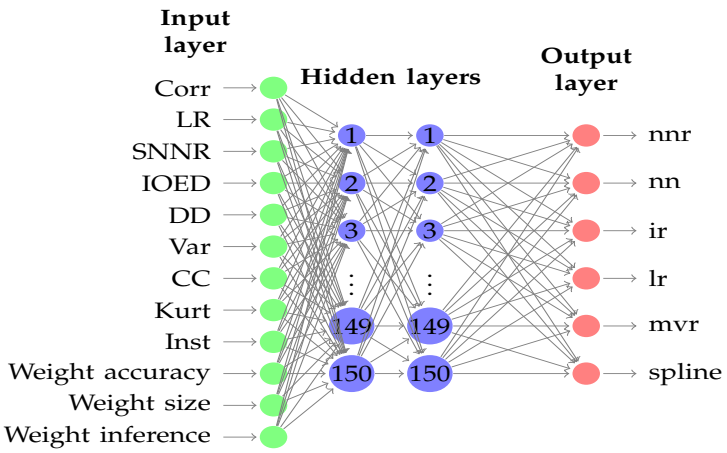


Fig. 6: Graphical representation of the meta-learner

	PC1	PC2
Corr	0.3721975	-0.3989694
LR	-0.39674234	0.39891286
SNNR	-0.31758717	-0.39124821
Var	0.3040892	-0.40151084
CC	-0.48117231	-0.19746618
Kurt	-0.0484191	0.32764965
Inst	0.34541101	0.42730057
DD	0.39692912	0.196145

TABLE 2: PC values of meta-features

The PCA shows the combined effect of the proposed meta-features, where the distributions are divided into four clusters with the use of both principal components. Combined with the correlation plot it indicates that the chosen meta-features explain different aspects of datasets.

5 Meta-Learner

This section explains the different aspects of the technical implementation of the meta-learner. We introduce our design choices and the reasoning behind them. The meta-learner is constructed as a neural network as the dimensionality of the meta-features is high. Also, the relations between the meta-features are not intuitive, but a neural network is able to deal with these relations. The neural network consists of an input layer, two hidden layers and an output layer. It is trained as a multiclass-classifier, where the output is the predicted class of the input. The input can be divided into two parts; meta-features and preferences of the user. The preferences of the user are three inputs, where each is a value indicating the importance of either accuracy of the model, inference time of the model, or size of the model.

The meta-learner is trained using the **categorical cross-entropy loss**, which is the standard loss function to use for multi-class classification, where only one class is applicable for each data point. This loss function trains the meta-learner to output a probability of the classes. Therefore, the output layer consists of six neurons, where each neuron represents a machine learning model type. These model types are presented individually later in Section 6. This is done through the use of the $softmax$ activation function in the output layer. The $softmax$ function normalizes the output so that it can be interpreted as probabilities. The hidden layers consists of 150 neurons each, with the $tanh$ and $relu$ activations functions respectively. The structure of the meta-learner can be seen in Figure 6. The meta-learner is trained on the datasets explained in Section 8.1. For each dataset meta-features are calculated, and performance for each model is measured. Then the models are ranked on the given dataset based on their performance and user preferences - this process is explained in Section 7.

6 Machine-Learning Models

The following section analyzes the different properties of different machine learning models as they excel in different settings. The ideal model displays low or no bias, such that we overfit the datasets as much as possible.
We have examined linear regression, multi-variate regression and neural network in our pre-thesis [10] which is why it will only be described brielfy in this paper.

6.1 K Nearest Neighbor Regression (KNN)

This non-parametric method is simple but effective in this environment. This method works by calculating the k nearest neighbours of a given point. Based on Figure 7 we want to predict the position of the red dot with an $x = 0.5$. Given a $k = 2$ we find the nearest points to be $(0.4, 5)(0.7, 7)$. Calculating the y value we take the average of y values of the k neighbours $y = \frac{5+7}{2} = 6$ [1].

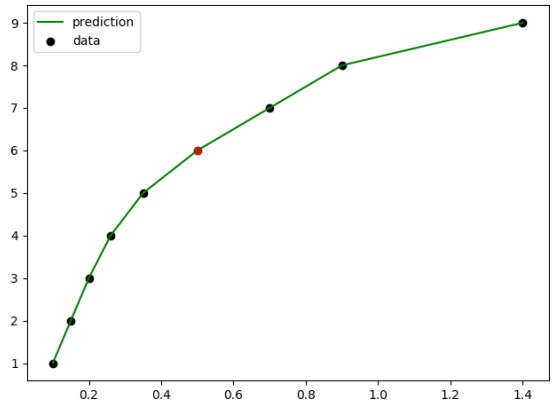


Fig. 7: K Nearest Neighbours $k = 2$

The difficult task is selecting a fitting k. We have the opportunity to force a low bias on this model and make it overfit with a low k. $K = 2$ is therefore selected. Another property this algorithm can exploit is that if all points are spread evenly along the y-axis, and relatively spread on the x-axis, such that we should choose the two adjacent data points on each side of the point we are trying to predict. Shortcomings of this model includes high computation cost on inference as this model is lazy. This means we have to compute the neighbors along with the average of the y-axis on inference. Furthermore, the model requires the knowledge of the entire dataset which increases the memory needed by the model.

6.2 Isotonic Regression (IR)

Isotonic regression is either a parametric method or a non-parametric method. The algorithm can as input take a weight for each observation to increase the bias. As we want a low bias we supply a weight of 1 utilizing unweighted isotonic regression which can be considered non-parametric. The algorithm has the constraint that we expect the data needs to be non-decreasing. Isotonic regression fits a free-form line to a dataset. We do this by minimizing the objective function $min \sum_i w_i(y_i - \hat{y}_i)$ where w is the weight. This is done by scanning the entire dataset and minimizing the objective function for each instance.
The downside to this method is that we also need to store the x-values for future linear interpolation of the predicted y value.

6.3 Spline Interpolation

A spline is a piecewise polynomial of quadratic- or cubic functions. Splines are parametric as it takes the number of knots used for interpolation as input. The number of knots indicates how many polynomials will be pieced together to interpolate data points. As seen in Equation 10, between each knot, we construct a quadratic equation and interpolate the x value if they follow the restrictions.

$$S(x) = \begin{cases} S_1(x) = a_1 + b_1 x + c_1 x^2, & x \in [x_1, x_2], \\ S_2(x) = a_2 + b_2 x + c_2 x^2, & x \in [x_2, x_3], \\ \vdots & \\ S_n(x) = a_n + b_n x + c_n x^2, & x \in [x_{n-1}, x_n], \end{cases} \tag{10}$$

An example of a spline can be seen in Figure 8, where three polynomial functions representing the spline.

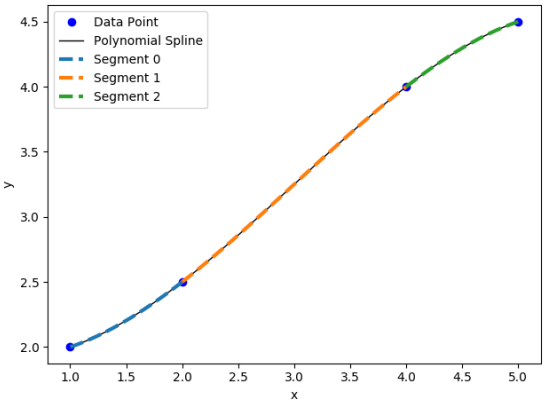


Fig. 8: Spline from three functions

The restriction of this algorithm is that the x values must be increasing, otherwise we will not be able to identify the function for the given data point.
The number of knots translates to the bias of the model. Many knots equals low bias. One downside to this model is the more knots the higher the memory consumption for the model, as we need to store the coefficients for the different functions as well as the placement of the knots.

Linear regression (LR) tries to find the best linear fit by calculating $y = ax + b$ and then finding the best values for a and b, such that given x, we find the correct value y.
Multi-variate regression (MVR) is an extension of linear regression which can, using multiple predictor variables, predict non-linear functions.
Neural networks (NN) can also be used for regression. Based on input features, it can predict the correct value for a given number of outputs. The neural network requires tuning in the form of number of hidden layers, number of neurons in each layer, choice of activation functions, etc.

Table 3 shows in what areas each machine learning model excels and also in which areas they have shortcomings.

The models that have a small memory size usually have shortcomings regarding accuracy. Models that have a

Model Name	Inference Time	Training Time	Memory Size	Average Prediction Accuracy
LR	Fastest	Fastest	Smallest	Low
MVR	Fast	Fast	Small	Low
NN	Slow	Slow	Largest	High
NNR	Fast	Fast	Large	High
IR	Fast	Fast	Large	High
SPLINE	Fast	Fast	Large	High

TABLE 3: Summation of machine learning models

high accuracy also has a high memory size. The table shows that a single optimal machine learning model is not possible, as there is some sort of tradeoff. Therefore it is required to rank the models in some other way.

7 Multi-Criteria Decision Analysis

The machine learning models can vary in size, inference speed and accuracy. Therefore, we investigate different multi-criteria ranking methods.
Making a decision on the best model is easy with only one criterion (accuracy for example), since we only need to choose the alternative with the best value for the given criterion. However, when the decision has to be based on multiple criteria several problems arise. These involve conflicting criteria, weights of the criteria and also the personal preference from the user [19].

Multi-Criteria Decision Analysis (MCDA) is a way to evaluate different criteria in the decision making state. Usually the MCDA methods evaluate the multiple criteria as either **cost** or **benefit**. Several MCDA methods exists, but no such thing as a super method [9]. Each method evaluates the criteria in different ways, and thus ranks the models differently. Therefore, it is important to choose a MCDA method that suits the problem in this paper. This includes a ranking of all the alternatives while also integrating the preferences of the user. These criteria of the MCDA means that a *compensatory* or a *partially-compensatory* MCDA method is needed. These methods allows compensations between different evaluations, meaning that a good performance on one criterion can counterbalance a poor performance on another [9]. This allows the user to determine which criteria are important for their dataset.

7.1 Weighted Sum Method

A MCDA method that allows compensatory ranking is the Weighted Sum Method. The most important aspect of this method is that users ability to weight the importance of the input. In ranking the models we look at three different input parameters: *Accuracy*, *Speed* and *Size*. An example of these values on a random dataset can be seen in Table 4.

The first step in the process is to normalize the values of the different attributes. As these have different scales we need to ensure that they are within the same scale. We employ *Linear Max* normalization as presented in [20]. In this process we identify which attributes describe

Model Name	*Accuracy* %	*Speed* **Sec**	*Size* **Byte**
IR	0.98	0.02	253520
MVR	0.54	0.05	392
LR	0.03	0.04	300

TABLE 4: Example evaluation of models

beneficial properties Bp and which are non-beneficial (cost) properties NBp. A general rule of thumb is to examine whether or not higher value will benefit the model. Therefore, $Bp = \{Accuracy\}$ whereas $NBp = \{Speed, Size\}$. Normalization of these are done as follows:

$$Normalized - Bp_i = \frac{Bp_i}{Bp_{max}} \tag{11}$$

$$Normalized - NBp_i = 1 - \frac{Bp_i}{Bp_{max}} \tag{12}$$

The normalized properties can be seen in Table 5.

Model Name	*Accuracy*	*Speed*	*Size*	*Score*	*Rank*
IR	1	0.6	0	0.528	1
MVR	0.55	0.0	0.9984	0.511	2
LR	0.03	0.2	0.9988	0.405	3

TABLE 5: Normalized model evaluation

Following this we calculate the score of the different models m by summing up the different normalized properties Np while adding a user specified weight w of importance for each property [21]. The weights in Table 5 are all 0.33.

$$score(m_i) = \sum_{i=1}^{N} (w_i * Np_i) \tag{13}$$

Lastly we rank the different models based on their scores as seen in Table 5.

7.2 Skyline Dominance

The Skyline query can also be used for multi-criteria ranking. The Skyline query is usually used in database context, and can easily be used in this context as well. A standard Skyline query optimizes two dimensions of a dataset, where these dimensions usually anticorrelate [5]. As an example, we want a model with high accuracy and low size as seen in Figure 9.

Name	Size	Inference Time	Accuracy
Lr	96	0.0000351	0.0001300
NNR	2219080	0.0007694	1.0000000
Ir	8865376	0.0001097	1.0000000
Mvr	424	0.0002053	0.0010562
Spline	12556008	0.0002222	1.0000000

TABLE 6: Skyline query results for a single dataset - optimizing three dimensions

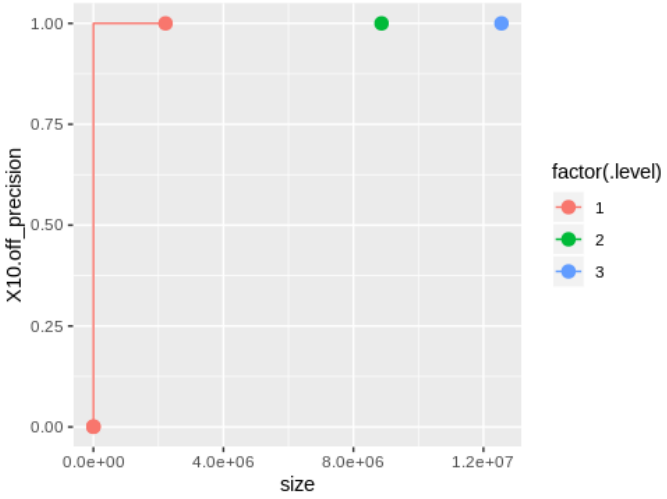


Fig. 9: Skyline query results for a single dataset - optimizing two dimensions

In this figure, each model has been giving a rank. Red is the best, green is second and blue the worst. In this example, it is not possible to choose between LR and NNR. LR has the lowest size of all, while NNR has the best accuracy. While there are other models, that have the same accuracy as NNR, they are outperformed by the size of NNR. The Skyline query are constructed using the Pareto composition. This means the result contains all models from a dataset which are not Pareto-dominated according to this preference. This means that we are not interested in those models, which are strictly worse in at least one dimension and worse/equal in other dimensions. However, our solution requires optimization for three dimensions; accuracy, size and inference speed. The Skyline query can also account for this. Choosing the best model based on a 3D graph can be hard, therefore a table with the results of a Skyline query optimizing three dimensions can be seen in Table 6.

In this table we see similar results as in Figure 9, where LR outperforms on size but has low accuracy. However, in this table inference time is also taken into account. This can be seen as IR being chosen over MVR, even though is it much larger.

The Skyline query is a powerful tool for ranking. However, in order to use Skyline as ranking method for our solution, it would require the meta-learner to predict the performance of a model. In our case this would mean predicting accuracy, size and inference time for each model, making it a very high dimensional regression problem. We consider this a much harder task than making a multi-class classifier, which is why the weighted sum method is chosen as ranking method.

8 Evaluation

In this section we present the datasets used for evaluation experiments. The meta-learner is then evaluated using machine learning parameters. Experiments was also conducted to show the performance on the RMI.

Using the meta-features from a dataset, the meta-learner can predict, based on the MCDA, which model to use on a given dataset. The model with the highest probability can then be chosen for RMI. An RMI was constructed and evaluated in our pre-thesis [10]. This RMI is used to test, whether the meta-learner can be used in such a setting. This RMI consists of two layers - a top layer with a single model, and a bottom layer consisting of several smaller models. The top model learns the entire distribution of the dataset, whereas the smaller models can be seen as 'experts' for a subset of the dataset. The meta-learner is tested on both the top layer and the bottom layer, where each model in the bottom layer might have varying distributions, meaning they should also be using different regression models.

8.1 Dataset

This section will present the datasets that will be used for the meta-knowledge base. The datasets chosen will vary in cardinality as this might have an impact on the algorithms presented in Section 6:

- *small* = 300−600
- *large* = 150.000−200.000

To have as varied data as possible we have chosen a combination of synthetic datasets and real world datasets. We draw samples from six different distributions which can be seen in Table 7.

Distribution name	Function	Variables
Uniform	$f(x) = \frac{1}{b-a}$	$a = 0, b = 1$
Beta	$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha-\beta)}$	$\alpha = 2, \beta = 2$
Wald	$P(x) = \sqrt{\frac{s}{2\pi x^3}} e^{\frac{-s(x-\sigma)^2}{2\sigma^2 x}}$	$s = 3, \sigma = 0.2$
Parteo	$p(x) = \frac{am^x}{x^{a+1}}$	a = 3
Lognormal	$p(x) = \frac{1}{\sigma x\sqrt{2\pi}} e^{(-\frac{(\ln x-\mu)^2}{2\sigma^2})}$	$\sigma = 1, \mu = 3$
Logistic	$P(x) = \frac{e^{-(x-\mu)/s}}{s(1+e^{-(x-\mu)/s})^2}$	$s = 2, \mu = 6$

TABLE 7: Synthetic distributions

Furthermore, we employ two real world datasets. The first is a collection of log entries from the Saskatchewan university website [16] with a cardinality of 2 million. The second is a sample from Open Street Map with a longitude ranging from $55.023-55.773$ with a cardinality

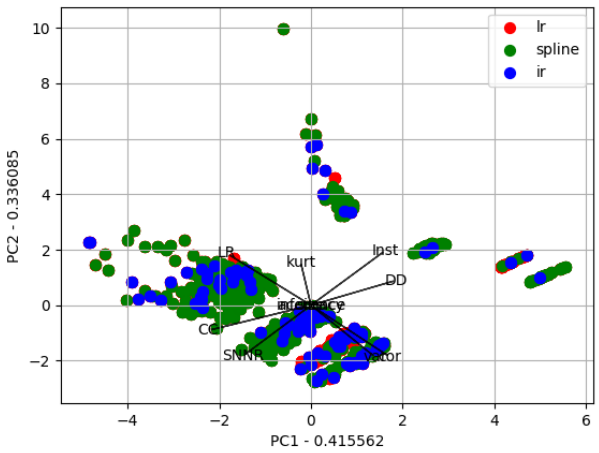


Fig. 10: Scatter plot with PCA values and model types

of 1.6 million. Examples of selected distributions can be seen in Figure 3.

8.2 Meta-Learner Evaluation

The meta-learner is evaluated through accuracy, meaning how often it predicts the correct model for a given dataset. The meta-learner is optimized using Kfold for parameter tuning. The meta-learner is evaluated on the test set. The training set contains 70% of the datasets, while the test set contains the last 30%. The datasets are shuffled in order to get better training sets and test sets. The meta-learner has a categorical accuracy of **74.21%**. The meta-learner was also evaluated on recall, precision and f1-score. The results of these measures can be seen in Table 8.

Recall	Precision	F1-Score
0.656746	0.808157	0.717845

TABLE 8: Measurement scores for the meta-learner

For all the measurements a score of 1 equals a perfect score, wheras the score of 0 is the worst score.
These measurements are another way of evaluating the meta-learner. Both recall and precision are based on an understanding of relevance. Precision scores the highest, which is a measure of how many predictions are relevant. This tells us that the meta-learner does not seem to predict that many false-positive predictions. The recall is lower, which means that the meta-learner is more prone to false-negative predictions. The F1-score is the harmonic mean between recall and precision, and is a good measure when there is no specific focus on either recall or precision. The F1-score has a value of 0.71, which is acceptable. A PCA plot was also made with the meta-features including the preferences of the user regarding accuracy, size and inference time. However, this time the data points represent the model types instead of the distributions. Figure 10 shows that the model types are not as evenly distributed as the distributions in Figure 5. All models are evenly distributed between the same 4 clusters, which also were prevalent in Figure 5. The principal components have a variance of $\approx 75\%$. This is another reason to use a neural network as the classifier, as the model types are not distinguishable from each other using the principal components.

8.3 Experiments on Learned Index

We evaluate The Adaptive Learned Index against the base model presented in The Case for Learned Index. We evaluate on the three parameters used for ranking *Accuracy*, *Size* in MB and *Time* in nano-seconds as presented in in Section 7. In this section, accuracy is referred to as *error rate*, as this is a better measure of the data structures' accuracy and time represents *inference time*.

Experimental Setting

The Python code used to run the experimental setup is available at https://github.com/timiane/adaptive-learned-index. The tests were conducted on the datasets presented in Section 8.1. The tests were run on a AMD 2600x Processor with 16 GB 3200 MHz ram.

Base Model

The base learned index model comes from [11] and our pre-thesis [10]. This model is not adaptive, in the sense that each layer contains the same machine learning models. Also, the base model relies on the user's knowledge of machine learning and data complexity. This means that if the user wants an optimal solution it requires a lot of trial and error in order to find the best fitting model that aligns with the user's preferences. In [11] and our pre-thesis [10] it was suggested that we either use a neural network or multi-variate regression as the top layer model, and then use linear regression as the bottom layer models.

Results

Table 9 shows the results of our experiments. The experiments were conducted by first testing the performance of the base learned index model with static machine learning models on the datasets. In this table the base model consists of a neural network model in the top layer and linear regression models in the bottom layer. The same tests were made with the adaptive learned index. The Adaptive Learned Index takes user preference into account, and therefore at least three tests were conducted each with a focus on either size, time or error rate. Lastly, an *optimal* solution test based on our knowledge of learned index was conducted. This means the top layer should have a preference on accuracy, as this will distribute the data to the bottom layer more evenly. The bottom layer should have a preference on size as these distributions usually are easier to learn.

Type	Config	Size(MB)	Time(ns)	Error Rate
Base Model	NN	47,13 (1.00x)	12646 (1.00x)	5.94 (1.00x)
Adaptive Model	Accuracy	8.10 (0.171x)	4310 (0.34x)	1.09 (0.18x)
	Size	4.56 (0.096x)	5347 (0.42x)	1.76 (0.30x)
	Time	5.32 (0.112x)	5350 (0.42x)	1.40 (0.24x)
	Optimal	4.57 (0.097x)	5257 (0.41x)	1.757 (0.30x)

TABLE 9: Adaptive learned index compared to the base learned index model

As the models with a high accuracy usually has a high size, the focus on size for the bottom layer is chosen to counter-balance this.

As seen in Table 9 the cells marked in green have the best performance in the category. In parenthesis we show the savings in size, inference time and error rate. It should be noted that the adaptive model with focus on accuracy has the lowest inference time. This is caused by the top layer model because it was predicted to be a SPLINE model. With more data and more models this problem should subside, as not enough data is provided for the meta-learner to correctly predict the most optimal solution based on the user's preference. The Adaptive Learned Index clearly outperforms the base model on all metrics. The preferences also seem to be taken into account, where the adaptive model chooses the models with the lowest error rate for accuracy preference, and the same goes for size preference. The optimal model does not have the best score in any of the metrics, however it has the second-best score on all the metrics except for error rate, which still has a good performance.

9 Conclusion & Future Work

We have shown that The Adaptive Learned Index improves the already exciting Case for Learned Index [11]. This was achieved by analyzing different features of complexity resulting in a vector describing the overall complexity of a dataset. We then ensured that these features provided information about different aspects of the datasets by analyzing their principal components and correlation to each other. Additionally, different machine learning models for regression were investigated. The performance of each model was evaluated based on different data distributions, such that each model was ranked accordingly using the Weighted Sum Method. This information was used to create a meta-learner that, given a complexity vector and user preferences on speed, size and accuracy, would predict the best performing machine learning model to learn the underlying CDF. The Adaptive Learned Index was tested on synthetic and real datasets providing results that showed this method improves the performance compared to the base structure presented in The Case For Learned Index [11].

Our results additionally showed that when it comes to user preferences and different distributions, different models should be chosen. The meta-learner aids the user in this decision, which can be difficult without exstensive knowledge about data complexity and machine learning models.

The Adaptive Learned Index presented in this paper is based on one-dimensional datasets. The Adaptive Learned Index could be expanded to include multi-dimensional data. This would require different meta-features to be extracted, which inspect class features along with expanding the model catalogue to be able to model the CDF accordingly. Further work could also include expanding the complexity evaluation of datasets. The features presented in Section 4 capture some complexity but expanding on these would deepen the understanding of data complexity. This would sharpen the predictions on what model to choose and thereby improving performance.

References

[1] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[2] M. F. Amasyali and O. K. Ersoy. "a study of meta learning for regression". 2009.

[3] A. Balte and N. N. Pise. "meta-learning with landmarking: A survey". 2014.

[4] N. Bhatt, A. Thakkar, and A. Ganatra. A survey and current research challenges in meta learning approaches based on dataset characteristics. *International Journal of soft computing and Engineering*, 2(10):234–247, 2012.

[5] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings 17th international conference on data engineering*, pages 421–430. IEEE, 2001.

[6] P. Brazdil and C. Giraud-Carrier. Metalearning and algorithm selection: progress, state of the art and introduction to the 2018 special issue, 2018.

[7] P. Brazdil and C. Soares. Ranking classification algorithms based on relevant performance information. *Meta-learning: Building automatic advice strategies for model selection and method combination*, 2000.

[8] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

[9] A. Guitouni and J.-M. Martel. Tentative guidelines to help choosing an appropriate mcda method. *European Journal of Operational Research*, 109(2):501–521, 1998.

[10] R. V. B. M. H. Jesper Hedegaard, Per Hedegaard Nielsen. Learned index and data complexity. Aalborg Universitet, 12 2018.

[11] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504. ACM, 2018.

[12] A. C. Lorena, A. I. Maciel, P. B. C. de Miranda, I. G. Costa, and R. B. C. Prudêncio. Data complexity meta-features for regression problems. *Machine Learning*, 107(1):209–246, Jan 2018.

[13] M. Magdon-Ismail and A. F. Atiya. Neural networks for density estimation. In *Advances in Neural Information Processing Systems*, pages 522–528, 1999.

[14] D. Oreski, S. Oreski, and B. Klicek. Effects of dataset characteristics on the performance of feature selection techniques. *Applied Soft Computing*, 52:109–119, 2017.

[15] D. Oreski, S. Oreski, and B. Klicek. Effects of dataset characteristics on the performance of feature selection techniques. *Applied Soft Computing*, 52:109 – 119, 2017.

[16] University of saskatchewan http request logs. ftp://ita.ee.lbl.gov/html/contrib/Sask-HTTP.html, 06 2019.
[17] M. Sewell. No free lunch theorem. http://www.no-free-lunch.org/, 03 2019.
[18] A. E. Smith and A. K. Mason. Cost estimation predictive modeling: Regression versus neural network. *The Engineering Economist*, 42(2):137–161, 1997.
[19] G.-H. Tzeng and J.-J. Huang. *Multiple attribute decision making: methods and applications*. Chapman and Hall/CRC, 2011.
[20] N. Vafaei, R. Ribeiro, and L. Camarinha-Matos. Normalization techniques for multi-criteria decision making: Analytical hierarchy process case study. volume 470, 04 2016.
[21] M. van Herwijnen. Weighted summation (wsum).
[22] J. Vanschoren. *Understanding Machine Learning Performance with Experiment Databases*. PhD thesis, Katholieke Universiteit, 05 2010.
[23] J. Vanschoren. Meta-learning: A survey. *CoRR*, abs/1810.03548, 2018.
[24] J. Zubek and D. M. Plewczynski. Complexity curve: a graphical measure of data complexity and classifier performance. *PeerJ Computer Science*, 2:e76, 2016.