A Certainty-Based Approach for Dynamic Hierarchical Classification of Product Order Satisfaction

Thomas Brink[a], Jim Leferink op Reinink[a], Mathilde Tans[a], Lourens Vale[a], Flavius Frasincar[a], Enzo Ido[a,*]

[a]*Erasmus University Rotterdam, P.O. Box 1738, 3000 DR, Rotterdam, the Netherlands*

Abstract

E-commerce companies collaborate with retailers to sell products via their platforms, making it increasingly important to preserve platform quality. In this paper, we contribute by introducing a novel method to predict the quality of product orders shortly after they are placed. By doing so, platforms can act fast to resolve bad quality orders and potentially prevent them from happening. This introduces a trade-off between accuracy and timeliness, as the sooner we predict, the less we know about the status of a product order and, hence, the lower the reliability. To deal with this, we introduce the Hierarchical Classification Over Time (HCOT) algorithm, which dynamically classifies product orders using top-down, non-mandatory leaf-node prediction. We enforce a blocking approach by proposing the Certainty-based Automated Thresholds (CAT) algorithm, which automatically computes optimal thresholds at each node. The resulting CAT-HCOT algorithm has the ability to provide both accurate and timely predictions by classifying a product order's quality on a daily basis if the classification reaches a predefined certainty. CAT-HCOT obtains a predictive accuracy of 94%. Furthermore, CAT-HCOT classifies 40% of product orders on the order date itself, 80% within five days after the order date, and 100% of product orders after 10 days.

Keywords: Dynamic hierarchical classification, Blocking approach, Threshold automation, E-commerce

1. Introduction

Since the late 1990s and especially during the COVID-19 pandemic, online shopping is gaining popularity as more and more consumers buy increasingly diversified products online [9, 31]. As a result, e-commerce companies often collaborate with retailers to sell products via their platforms, making it increasingly important to preserve platform quality. That is, a platform and its retailers must be reliable and deliver products and service of high quality in order to maintain the platform's reputation [13]. Therefore, e-commerce platforms are currently concerned with product order quality to perfect the products and services that they provide. We define the *product order quality* as the extent to which platforms are satisfied with an order. A product order is defined as a *Perfect Product Order* if the order is not cancelled, the product is delivered on

*Corresponding author; tel: +31 (0)10 408 1340; fax: +31 (0)10 408 9162

Email addresses: `455893tb@student.eur.nl` (Thomas Brink), `452358jl@student.eur.nl` (Jim Leferink op Reinink), `446685mt@student.eur.nl` (Mathilde Tans), `447398lv@student.eur.nl` (Lourens Vale), `frasincar@ese.eur.nl` (Flavius Frasincar), `534787ei@eur.nl` (Enzo Ido)

time, the customer does not file a case, and the product is not returned. However, if any of these aspects goes wrong (e.g., the product is not delivered on time), we define this as a *Deficient Product Order*. Depending on the type and number of aspects that go wrong, a Deficient Product Order can be classified as Mildly, Medium, or Heavily Deficient. Finally, if not all information on product delivery is known and no other aspects go wrong, the order is labelled as an *Inconclusive Product Order*. The split of an order's quality into the Perfect, Inconclusive, and Deficient classes is defined as the *general classification*, while the split of product order quality into the Perfect, Inconclusive, and three gradations of Deficient classes is defined as the *detailed classification*. Whereas assessing the quality of product orders usually takes multiple weeks, ideally, e-commerce platforms should be able to detect the quality of an order directly on or shortly after the order date - the date the order was placed. For example, if a retailer in the platform consistently delivers products that are being returned, the platform wants to flag this retailer as quickly as possible and, if necessary, take measures to preserve platform quality. Therefore, in this paper, we investigate the following research question:

How can we accurately predict the quality of product orders shortly after the orders have been placed?

To answer this question, we predict the quality of product orders for a large e-commerce platform in the Netherlands at different points in time after an order is placed and evaluate the quality and timeliness of our predictions. We refer to the labels that are assigned to product orders based on their quality as *class labels*. We specifically focus on assigning such class labels to product orders in a fast and accurate way, such that the company, synonymous to 'e-commerce platform' and 'platform', is able to detect any problems quickly and confidently. However, we do not focus on the drivers of those class labels, that is, the reasons why a product order has a particular class label. These drivers, which we refer to as *determinants*, include cancellations, late deliveries, customer cases, and returns.

Clearly, the longer the period between the order date and the time of prediction, the more information will be available about the status of the product order, such as any of the four product order determinants or transporter information, and the more accurate we expect our prediction to be. On the other hand, the longer the period between the order date and the time of prediction, the less actionable our prediction will be for the platform. For example, if a retailer consistently delivers products for which customer cases are started, the company should know about this issue as soon as possible. We refer to this aspect as the *timeliness* of our predictions. Hence, overall, we face a trade-off between accuracy and timeliness.

In this paper, we deal with the previously identified trade-off by developing a novel hierarchical classification algorithm to predict the quality of product orders. In hierarchical classification (HC), the hierarchical structure of the classes to be predicted is explicitly taken into account, which may improve classification performance as opposed to a 'flat' classification approach, which does not consider this hierarchical structure [22]. Given that the problem at hand has a clear and logical hierarchy, it makes sense to use HC in this research as opposed to a 'flat' classification approach. HC organises classes into a tree-based hierarchy, which in our case combines the general and detailed classifications into one tree that captures the full hierarchical structure of the class labels. The leaf nodes in our tree, which form the main focus of our predictions, are represented by the detailed classification labels.

To be able to predict leaf nodes while taking into account the trade-off between accuracy and timeliness, we introduce the Hierarchical Classification Over Time (HCOT) algorithm, which dynamically predicts instances by means of non-mandatory leaf-node prediction in an iterative fashion. That is, when we are not 'certain' enough about classifying a certain instance on a specific day, we block this instance in the hierarchy and try to assign a more detailed label to this instance the next day when, possibly, new information becomes available. In order to incorporate the concept of certainty, we develop the Certainty-based Automated Thresholds (CAT) algorithm, which automatically computes optimal probability thresholds per node in the hierarchy that an instance must exceed to continue to a deeper level in the tree. If an instance passes the thresholds on a specific day, we are certain enough about the classification of this instance and thus classify it according to the label corresponding with the leaf node that is reached. By combining the HCOT algorithm with thresholds computed by the CAT algorithm, we develop a classification procedure (CAT-HCOT) that is able to classify product orders with certainty over time, thus capturing the trade-off between accuracy and timeliness.

The relevance of this paper is threefold. First, we are, to the best of our knowledge, the first to apply hierarchical classification over time. Second, we contribute to the existing literature by introducing the use of the CAT algorithm to deal with uncertainty in classification problems. Third, in contrast to most of the existing literature, we apply hierarchical classification to a shallow (3-level) tree with strongly imbalanced data that is tailored to the domain of product order satisfaction in e-commerce.

We find that CAT-HCOT provides predictions with a global accuracy of 94%. Moreover, global and class-specific hierarchical precision, recall, and F_1-score almost always achieve values above 90%. Thus, CAT-HCOT exhibits strong predictive performance. With respect to timeliness, CAT-HCOT is able to classify around 40% of orders on the order date itself, approximately 80% of orders within 5 days after the order date, and, by design, 100% of orders after 10 days. All in all, we find that CAT-HCOT adequately captures the accuracy versus timeliness trade-off by using a certainty-based classification approach, while a static method does not allow for such a balanced trade-off. We further find that CAT-HCOT achieves major improvements over a flat baseline method in terms of hierarchical precision and recall scores on the most specific and least occurring Deficient classes, which illustrates that taking into account the hierarchical class structure improves predictive performance and allows for dealing with imbalanced data.

This paper is structured as follows. In Section 2, we review the literature on hierarchical classification, where we focus on the most prominent techniques, the non-mandatory leaf-node prediction blocking approach, hierarchical performance measures, and application domains. In Section 3, we present our data set along with overall descriptive statistics. Then, in Section 4, we outline the directions of our research and the methods used. In Section 5, we present our results. Last, in Section 6, we draw our conclusions and discuss interesting directions for future work.

2. Related Work

In this paper, we apply hierarchical classification in the area of company satisfaction in e-commerce, which is, to the best of our knowledge, a rather novel area. One work in a similar domain is performed by Vandic et al. [26], who propose a framework for hierarchical product classification in e-commerce. More frequent is the classification of product order satisfaction from the customer's perspective as by Zheng et al. [30] or Jiang et al. [15], for example, who consider the classification of customer reviews. Due to their text-based nature, customer reviews are also suitable for hierarchical classification [21]. In fact, most research on hierarchical classification focuses on the area of text categorisation [12]. However, the application of hierarchical classification to the quality and satisfaction of product orders from a company perspective has not been previously explored.

In the remainder of this section, we outline our theoretical framework for hierarchical classifications. We elaborate on the general concept, after which we highlight the techniques of non-mandatory leaf-node predictions. Subsequently, we discuss hierarchical classification performance measures and, after we have introduced the general concepts, we set up the context of the relevance of our work.

2.1. Hierarchical Classification Techniques

Hierarchical classification (HC) is defined as a classification approach where the classes to be predicted are organised into a class hierarchy [22]. According to Freitas and Carvalho [12] and Sun and Lim [24], HC classification methods differ on a number of criteria. The first criterion is the type of hierarchical structure that is used, which can be either a tree or Directed Acyclic Graph. In the latter, a node can have more than one parent node, which is not possible in the former. The second criterion makes a distinction between whether or not classifications are required to reach a leaf node. More specifically, *mandatory leaf-node prediction* (MLNP) forces the HC algorithm to always assign an instance to a leaf node, whereas *non-mandatory leaf-node prediction* (NMLNP) allows the algorithm to stop the classification in any node at any level of the hierarchy. Finally, the third criterion is related to how the hierarchical structure is explored. Silla and Freitas [22] identify two relevant approaches: flat classification and local classification.

The flat classification (FC) approach is the simplest hierarchical classification technique and ignores the class hierarchy—it typically only predicts the classes at the leaf nodes. Therefore, a flat classifier is equivalent to a traditional classifier during training and testing, and does not explore parent-child class relations. However, assignment of an instance to a particular leaf class still implies that this instance also belongs to all ancestor classes based on the original hierarchical structure [22].

The local classification (LC) approach takes into account the class hierarchy by using a local information perspective. The three standard approaches to classifying locally are *local classification per node* (LCN), *local classification per parent node* (LCPN), and *local classification per level* (LCL) [22]. We prefer LCPN and LCL over LCN, since these methods require fewer classifiers and do not need a specific strategy for selecting training examples [22]. Furthermore, all LC algorithms use a *top-down approach* in their testing phase. This means that for each test instance, they first predict its first-level, and then use this predicted class to determine which classes are considered at the second level, and so on, until they make the most specific prediction possible. A disadvantage of this top-down prediction approach is the occurrence of *error propagation*, where prediction errors at any class level are propagated downwards through the hierarchy. One way to reduce or stop this error propagation is to use a *blocking approach*, which blocks the process of passing down instances in the hierarchy if the prediction at the current level is not accurate enough according to some criterion, such as a 90% predicted probability requirement. A downside of this blocking approach is that the user might end up with less specific, and therefore less useful, class predictions.

In hierarchical classification, similar classification methods can be used to those applied to non-hierarchical (multi-class) classification, albeit that different classifiers can be used at different levels. For example, in LCPN, the choice of classification algorithm can be made per parent node. In the existing literature, examples include Naïve Bayes [26], Logistic Regression [14], Support Vector Machine [10], Decision Tree [6], Random Forest [8], and Neural Network [28].

2.2. Blocking Techniques

Among the various blocking techniques, we opt to implement thresholds at each node. Instances allocated to a node are only accepted if their predicted probability exceeds a threshold. To determine the thresholds, we distinguish two possible strategies. Firstly, a threshold can be set manually, for instance, to block instances for which the predicted probability is less than 80%. The second option is to select an optimal threshold automatically. This means that the threshold is determined during the training phase to optimise a specific quantity, such as the accuracy [5] or the F_1-score [1, 4]. In this case, the thresholds can be considered hyperparameters to the classification algorithm. A threshold selection procedure that plays into an accuracy versus timeliness trade-off such as ours has not been explored before.

2.3. Hierarchical Performance Measures

Sokolova and Lapalme [23] define *hierarchical performance measures* that explicitly incorporate the class hierarchy to evaluate predictive performance. Costa et al. [7] discuss several distance-based, depth-dependent, and semantics-based hierarchical performance measures. These often depend on choices specified by the user, which makes them rather subjective. Therefore, Sokolova and Lapalme [23] and Silla and Freitas [22] recommend using hierarchy-based measures instead, which use the concepts of ancestral and descendent classes. Kiritchenko et al. [17] suggest hierarchical extensions of 'flat' performance measures. More specifically, they define hierarchical precision (hP), hierarchical recall (hR), and hierarchical F_1-score (hF_1). The big advantage of these measures is that they can be effectively applied to any HC scenario—whether the problem is tree-structured or DAG-structured, and MLNP or NMLNP.

A complexity arises in NMLNPs due to blocking. Instances with a specific true class may not reach this class if they have been blocked after a higher-level classifier. Consider, for example, a text categorisation exercise where a document truly belongs to the class 'physics' but is blocked at the class 'science'. While the classification 'science' is correct, it is not all-encompassing. Sun et al. [25] propose a measure to keep track of the proportion of test instances that is blocked by a certain classifier, despite belonging to a more specific class in the hierarchy. The authors define this measure as the *blocking factor*. We adapt the blocking factor in our research and propose a novel threshold-based approach to decide which instances to block.

2.4. Relevant Context

In most applications of hierarchical classification, the class hierarchy is quite large in terms of the number of nodes. Silla and Freitas [22] report that most hierarchical classification approaches clearly outperform flat classification approaches, because the former explicitly accounts for the structure in the class hierarchy. However, one can imagine that this predictive advantage becomes less pronounced as the class hierarchy becomes smaller, thereby approaching the most simple flat hierarchy. In this research, we use a rather small class hierarchy, consisting of only seven nodes (internal and leaf), whereas the vast majority of class hierarchies in existing applications consist of ten or (much) more nodes [22].

As for the applications of NMLNP, the choice of blocking instances from reaching leaf nodes in the hierarchy often rests on instances not belonging to a leaf node, but rather to an internal class, e.g., in document text classification [4]. However, in our approach, NMLNP plays a different role: that of ensuring 'certainty' of predictions over time. Namely, if an instance gets blocked at a certain point in time, we try to predict it the next day, making use of the newly available information, e.g., product was delivered late. Although making predictions over time has been subject to previous research [19], applying this concept to hierarchical classification has, to the best of our knowledge, not yet been investigated.

Lastly, our data is skewed across classes, which means that we face an imbalanced data problem in our hierarchical classification task. A common approach to deal with imbalanced data in classification is that of cost-sensitive learning, where larger costs are assigned to misclassifications of less represented classes [11]. As Zheng and Zhao [29] show, such cost-sensitive approaches can drastically improve the performance of hierarchical classification algorithms. The simplest approach herein is to assign class weights inversely proportional to the rate of occurrence in the data set, which is the approach we will take in this research.

3. Data

In this section, we discuss our process of data cleaning and validation, in addition to presenting important descriptive statistics. Our data set contains roughly 4.8 million observations on product orders at a large e-commerce platform in the Netherlands from January 2019 to December 2020.

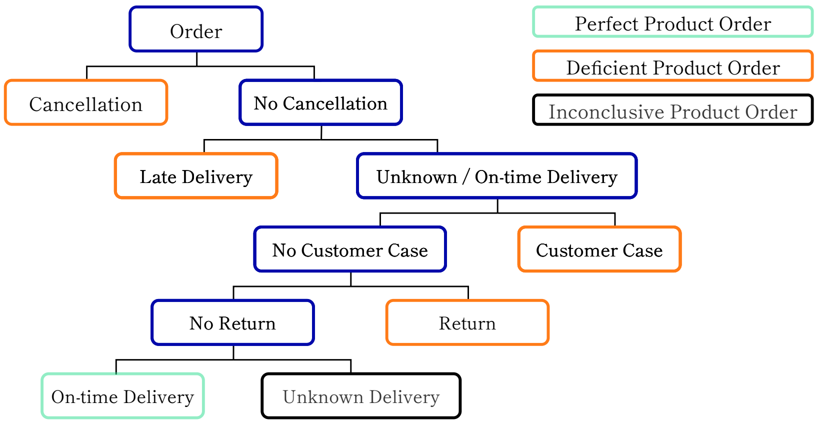


Fig. 1. Class label decision tree provided by the company.

3.1. Data Description

We predict the product order quality at multiple points in time after the order date. To make this possible, each product order is included in the dataset several times. The difference between these observations is the amount of information on the current status of the product order. This mainly includes information on the product order determinants, but you can also think of information about the transporter, the product itself or the seller. Typically, all this information is unknown or limited on the order date itself and only becomes available in the period afterwards. For example, on the day of ordering we probably do not observe any information on delivery or the transporter company, while a few days later, this information probably is known. Furthermore, during a process of an order we gain more knowledge about the quality of similar (completed) product orders, such as the fraction of deficient orders. Clearly, the more information we have, the more certain we can be about the predictions. Once all information on the determinants corresponding to a product order is known, one can determine the corresponding class label with respect to the tree in Figure 1, representing the platform's business logic.

As determined by the company, cancellations are only allowed between 10 days after the order date and the other determinants can take up to 30 days. As such, the platform currently finalises the classifications of product order quality only after the 30 days have passed. Anywhere before this period, there still might exists uncertainty about some determinants, such that prediction is necessary. This is showed clearly in Figure 2. From this figure we can derive the percentage of product orders that has information available on a specific determinant per day after the order date. For example, we see that after 10 days we know for 76% of the orders whether a return was requested or a case created. Information on on-time delivery does not further increase after 10 days since for a large fraction of orders the delivery status is not traced and remains unknown forever. For all determinants, relatively little extra information becomes available in the last 20 days. Since companies have the urge to classify orders as soon as possible, we set a maximum on the classification period. Therefore, in this research, we use a time window of 11 days to predict the product

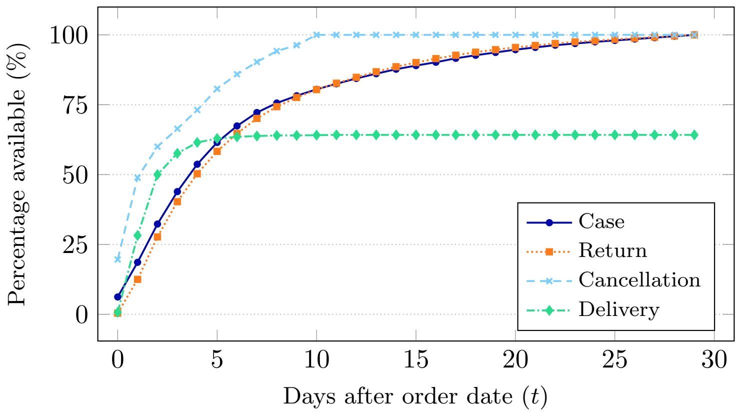


Fig. 2. Availability of determinant information 0 to 30 days after the order date.

order quality, that is, on the order date itself (day 0) and in the 10 days afterwards (day 1 to 10)[1]. As target variable we use the finalised classifications by the company.

3.2. Data Procedures

As the data set contains some noisy observations, we first clean the data set. Noisy observations include product orders with conflicting characteristics, such as a return registered before the order has been placed. In nearly all types of noise, we decide to remove the respective instances from the data set, which leads to 2,624 product orders being removed. In the case of product orders for which more items are returned than originally ordered, we decide to manually modify those observations; we adjust the quantity returned to match the quantity ordered, which affects 14,722 product orders. Our motivation for this approach is twofold. First, we remark that the ordered quantities are directly related to the money flow into the company. If an error had been produced here, we expect that it would have been detected at an earlier stage. Second, the modifications are minimal. In nearly 90% of the modifications, the adjustment is by 1 unit.

Last, in order to make our data set well-suited for prediction models, we select, transform, and create (new) variables. Most importantly, we create dummy variables corresponding to the determinants, which indicate whether or not the information on the determinant is known to the company at a specific moment. For example, we create a dummy variable for product delivery. If the information on late product delivery becomes available 2 days after the order date, the dummy variable takes value 0 in these first 2 days, and then takes value 1 as soon as the information is known. If the product is delivered on time, there is no late delivery at all and the dummy variable keeps value 0 for all observations of this order. Furthermore, we create variables related to the transporter, retailer, and product ordered. For example, we have a variable containing the fraction of deficient product orders over the last few days for the retailer related to the order. In case we suddenly observe a large increase it is more likely that the current order also ends up deficient.

[1]We opt for 11 days because it balances timeliness and information availability. Extending this classification period goes against the preference of quick classifications with the goal of acting on bad quality product orders. On the other hand, when shortening this period we possibly have too little information available to make a valid classification.

Table 1
Occurrence of determinants as percentage of total number of product orders.

	No cancellation	On-time delivery	No customer case	No return
True	99.50%	60.75%	96.56%	94.12%
False	0.50%	3.47%	3.44%	5.88%
Unknown	N.A.	35.78%	N.A.	N.A.

From Figure 1 one can determine the class label once all the determinants corresponding to a product order are known. Because of this property, the determinant dummy variables can be regarded as 'near-perfect predictors' for the class labels. Overall, as more and more days after the order date pass, more information on the status of the product order is known. As a result of this construction, on each point in time after the order date, we can train a prediction model with updated information on the status of the product order.

3.3. Descriptive Statistics

In this section, we discuss the most important descriptive statistics of our (cleaned) data set, consisting of 4.8 million observations. The majority of product orders is a Perfect Product Order (56%), about one-third is an Inconclusive Product Order (32%), and the minority is an Deficient Product Order (12%). Using the detailed classification system, we find that the majority of deficient product orders is a Mildly Deficient Product Order (9%), and that a small fraction of all product orders is a Medium Deficient Product Order (2%) or a Heavily Deficient Product Order (1%). Thus, the data is obviously skewed with respect to the different class labels. Hence, in our classification task, we face an imbalanced data problem.

Table 1 gives us more insight in the values of the determinant variables—that is, the occurrences of cancellations, late deliveries, customer cases, and returns. We find that 0.5% of product orders are cancelled, 3.4% of orders have a customer case related to it, and 5.9% of orders are returned. In addition, 3.5% of product orders are delivered late, while roughly 36% of product deliveries are unknown.

4. Methodology

In this section, we introduce our hierarchical classification approach that classifies instances over time using an automated threshold procedure. We first provide some general notation and definitions, after which we present the structure of the tree-based class hierarchy. Next, we discuss the main components of HCOT as well as classifier selection and optimisation. Then, we present the CAT algorithm, discuss the testing procedure using the CAT-HCOT algorithm, and evaluate our model. Last, we outline two baseline methods to compare our model with and discuss the software used.

4.1. Notation and Definitions

In this research, we use a tree-based structure to organise our classes into a hierarchy. Following the definitions provided by Silla and Freitas [22], a *tree-based class hierarchy* is defined as $\mathcal{H}$, containing the finite set of nodes $\mathcal{N}$. We can classify a node $m \in \mathcal{N}$ according to different tree-based terminology systems:

- **Root/standard**. We define R as the root node of the hierarchy and $\mathcal{S} \subset \mathcal{N}$ as the set of all standard nodes, where $\mathcal{S} = \mathcal{N} \setminus \{R\}$.
- **Internal/leaf**. We define $\mathcal{I} \subset \mathcal{N}$ as the set of internal nodes and $\mathcal{L} \subset \mathcal{N}$ as the set of leaf (or external) nodes, where $\mathcal{I} \cup \mathcal{L} = \mathcal{N}$.
- **Parent/child**. We define $\mathcal{C}_m$ as the set of all child nodes of node m and $\mathcal{S}_m$ as the set of all sibling nodes of node m. Furthermore, we define p_m as the parent node of node m.
- **Ancestor/descendant.** We define $\mathcal{A}(m)$ as the set containing node $m \in \mathcal{N}$ and all its ancestor nodes, and $\mathcal{D}(m)$ as the set containing node $m \in \mathcal{N}$ and all its descendant nodes.

A training instance is defined as $u \in \mathcal{U}$, where $\mathcal{U}$ is the set of all training instances and $|\mathcal{U}| = M$. We assign a test instance $r \in \mathcal{R}$ to a class with respect to the hierarchy, where $\mathcal{R}$ is the set of all test instances and $|\mathcal{R}| = N$. By definition of the tree-based hierarchical structure, an instance assigned to a particular class naturally belongs to all its ancestor classes. We define the i-th test instance by $r_i = (\mathbf{x}_i, l_i) \in \mathcal{R}$, where $\mathbf{x}_i \in \mathbb{R}^D$ is the vector of values on the D model features and $l_i \in \mathcal{L}$ is the true label of instance i.

4.2. Tree-Based Class Hierarchy Structure

In order to use our hierarchical classification algorithm, we define a tree-based class hierarchy structure based on business logic. We do not use a DAG structure, since, as a consequence of business logic, any node in the tree cannot have more than one parent node. Figure 3 shows the 3-level tree representing the class hierarchy: first, product orders are classified either as Inconclusive (<0>) or Conclusive (<1>). If the order is classified as Inconclusive, it means that we expect all service criteria to be met, but it is unknown whether or not the product is delivered on time. If the order is classified as Conclusive, it means that we can predict the quality of the order as either Perfect (<1.1>) or Deficient (<1.2>). The order is classified as Perfect when we expect all service criteria to be met, that is, no cancellation, on-time delivery, no return and no case. In contrast, the order is classified as Deficient when at least one service criterion is not met. If the order is classified as Deficient, we can assign it to a more specific Deficient class. Based on the extent to which the product order quality is lacking, we classify the order either as Mildly Deficient (<1.2.1>), Medium Deficient (<1.2.2>), or Heavily Deficient (<1.2.3>). Since the exact business rules for determining the degree of deficiency are company-specific and rather complex, we do not discuss them in this paper.

4.3. Hierarchical Classification Over Time (HCOT)

In our problem setting, we face a trade-off between accuracy and timeliness. On the one hand, product orders should be classified as accurately as possible to ensure reliable predictions, but, on the other hand, orders should be classified as soon as possible to ensure actionable predictions. To handle this trade-off, we use hierarchical classification of order satisfaction for $t = 0, 1, \ldots, 10$ days after the order date. We refer to this approach as *hierarchical classification over time.* After 10 days, each product order should be assigned to a label which corresponds to a leaf node $l \in \mathcal{L}$. An order can have at most one label per hierarchy level. For example, an order cannot be classified as both Perfect (<1.1>) and Deficient (<1.2>). Therefore, any instance can be assigned only one path of predicted labels, which means that HCOT makes use of *single path prediction*. The HCOT algorithm consists of three main procedures: training, testing, and blocking.

Training procedure. In HCOT, we train the hierarchy $\mathcal{H}_t$ independently per day using the available information on day t. This implies that each 'daily' hierarchy $\mathcal{H}_t$ is trained on the same (number of) instances and the same features, while the information contained in those features may change over time as more information comes in. For each daily hierarchy, we make use of LCPN, which means that we train a classifier at each parent node. The parent nodes in our class hierarchy (indicated by the dotted lines in Figure 3) include the root node <R>, node <1> (Conclusive), and node <1.2> (Deficient). At <R> and <1>, we need a binary classifier, as both nodes have only two child nodes. At <1.2> we need a multi-class classifier, as this node has three child classes. Since each level has only one parent node in our hierarchy, we note that LCPN is equivalent to LCL. We train the classifiers using the true training labels, i.e., we train the classifier at <R> with all training data, the classifier at <1> with all instances that are truly Conclusive, and the classifier at <1.2> with all instances that are truly Deficient. In Section 4.4, we elaborate on the way we select our classifier methods and optimise the corresponding models.

Testing procedure. Next, by applying the trained classifiers, we can assign test instances $r_i \in \mathcal{R}$ using HCOT. We do so by means of iterative top-down class-prediction over time with NMLNP, which implies that a test instance is not necessarily assigned to a leaf node $m \in \mathcal{L}$ (e.g., <1.2.1>) on a certain day, but can also get blocked at an internal node $m \in \mathcal{I}$ (e.g., <1.2>). In general, for day $t = 0, 1, \ldots, 9$, an instance first goes through the hierarchy $\mathcal{H}_t$ and, if it is not classified into a leaf node, is sent to $\mathcal{H}_{t+1}$. Each day, starting from the root node <R>, we first try to assign an instance to a label at the first level and then use that label to determine whether we continue to assign the instance to a label at a lower level. Hence, although an instance can be classified at an internal node on a given day, the next day we start the classification process again from the top of the tree. This overcomes the issue of *error-propagation*, which means that prediction error at any class level is propagated downwards with respect to the hierarchy. This is a clear disadvantage of the

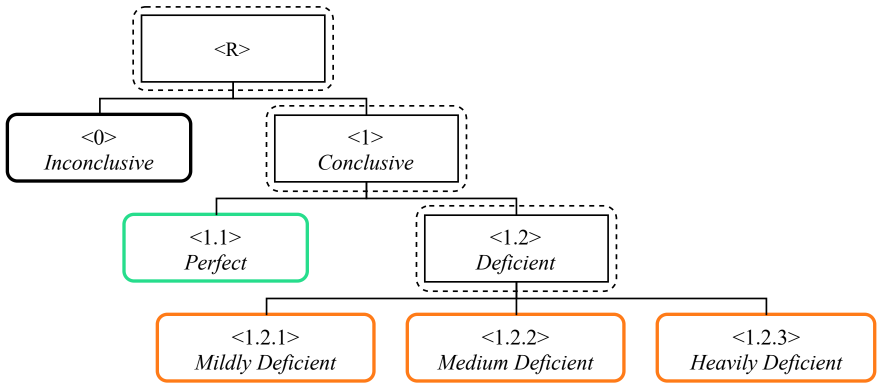


Fig. 3. Pre-defined class hierarchy based on business logic.

top-down class-prediction procedure. By re-starting the classification process again, we avoid propagation of possible errors made on the previous day. If an instance is still not classified to a leaf node on day 9, we make sure this instance is classified into a leaf node on day 10. This way, all instances are assigned to a leaf node within 10 days after the order date.

Blocking procedure. We use thresholds as the blocking approach to handle NMLNP and determine whether a test instance continues to the next level in the tree on a certain day. The thresholds are implemented at each node $m \in \mathcal{S}$: if the probability for an instance to reach a given child node does not exceed its threshold, e.g., at least a predicted probability of 80%, then the classification is blocked at the corresponding parent node. When an instance does not get blocked, it reaches a leaf node and gets classified. We refer to this process as *leaf-classification*. Since this instance passes all thresholds on its path, we are confident enough about our classification and thus expect to end up with accurate predictions. In addition, since our testing approach implies that all instances go through the hierarchy in an iterative fashion (over time), we make sure to classify each instance as soon as possible. Thus, the blocking approach in HCOT allows us to face the trade-off between accuracy and timeliness. In Section 4.5, we propose and discuss the algorithm for computing certainty-based automated thresholds.

In short, HCOT works as follows. Each day, starting on the day of ordering $t = 0$, an instance can either be assigned to a leaf class or be blocked at one of the parent (internal) nodes. If an instance gets classified into a leaf class, we consider its classification as final and we do not try to classify the instance again the next day, as we assume that a platform wants to immediately act on a predicted product order quality. Hence, from a business perspective, there is no need to classify a leaf-classified instance again on the next day [16]. On the other hand, if an instance gets blocked at an internal node, there is insufficient information available for the instance to be assigned to a leaf class and, as a result, we try to classify the same instance again the next day, starting over again from the root node of the hierarchy. Still, by assigning internal labels, HCOT provides actionable insights even if there is not enough certainty of providing leaf-node predictions. Because we want all instances to be classified within 10 days, we drop our blocking approach from HCOT on day 10, such that all not-yet-classified instances get assigned a leaf class on day 10 (using MLNP).

Figure 4 graphically illustrates the process of HCOT with an example. On day 0, an instance gets

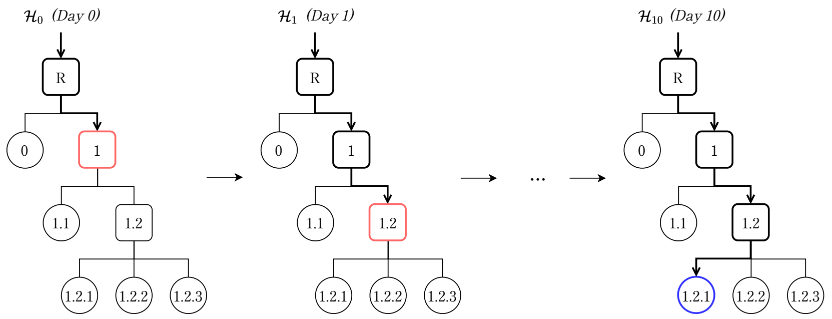


Fig. 4. Graphical representation of HCOT with $\mathcal{H}_0, \ldots, \mathcal{H}_{10}$. Red colour (light-grey in B&W printing) indicates blocking at the corresponding parent node. Blue colour (dark-grey in B&W printing) indicates leaf-node classification.

assigned to node <1>, but is blocked at this node due to a lack of certainty. On day 1, we try to leaf-classify the instance again. This time, the instance gets assigned to node <1.2>, but is again blocked from further hierarchical classification. In this specific example, the instance gets blocked up until day 9, such that we use MLNP on day 10 to ensure that the instance is eventually assigned to a leaf class, in this case <1.2.1>.

4.4. Classifier Selection and Optimisation

In HCOT, we construct our 'optimal' hierarchical classifier by optimising each daily hierarchical classifier ($\mathcal{H}_t$) independently. To do so, there are three components we take into account: classifier selection, imbalanced data, and model tuning. All three components are applied to daily hierarchies, which are eventually put together over time to form our HCOT classifier. First, we consider the process of selecting classifiers. As noted, we train each $\mathcal{H}_t$ using LCPN, implying that we train classifiers per parent node. Therefore, different classifiers can be used at different parent nodes in the hierarchy. In addition to providing flexibility in classifier selection, this could also lead to improved predictions. Namely, different algorithms may prove to be better able to differentiate between observations in different 'regions' (parent-child splits) of our data. Therefore, each day, we optimise the combination between classifiers at the different parent nodes. To optimise such combinations, we perform an exhaustive search across all possibilities. Since our tree consists of three parent nodes (see Figure 3), considering K classifiers leads to evaluating $C = K^3$ combinations. To select our classification methods, it must be noted that our blocking approach requires probabilistic class predictions. The first classifier we choose is the Random Forest (RF) classifier, which, through averaging classifications over various decision trees, is known to possess strong predictive performance and provide specific class probabilities for instances. The second classifier we select is Logistic Regression (LR), which is a probabilistic classification method by nature. Taking these $K = 2$ classifiers, we evaluate $C = 2^3 = 8$ combinations on a daily basis. Clearly, C explodes with K, such that we decide to keep K small. However, in case more classifiers would be selected, we could use (greedy) heuristics to overcome the computational difficulty of an exhaustive search and still arrive at a near-optimal solution.

Second, in training the combinations of the above two classification methods for each $t = 0, 1, \ldots, 10$, we face an imbalanced data problem. That is, the distribution of data across the child nodes of all parent nodes in our hierarchy is skewed. If we define the class distribution as the relative occurrence of instances per child node in the data set, we find that the class distributions are roughly 2:1, 5:1, and 7:2:1, for the root parent node <R>, parent node <1>, and parent node <1.2>, respectively. Class imbalance may be a problem, as it biases classifiers towards predicting the most occurring classes [29]. In our application, we value instances from each class equally important and, therefore, apply a cost-sensitive imbalance approach at each parent node. This approach assigns larger weights to classifying instances from less prevalent classes. That is, assigned weights to classes are inversely proportional to the rate of occurrence of these classes. In this way, classifiers at each parent node are less biased towards assigning the most prevalent labels.

Third, having selected the combinations of classifiers to consider and accounting for imbalanced data, we turn to tuning each $\mathcal{H}_t$. This implies that, for each t, we need to find the best combination of classifiers

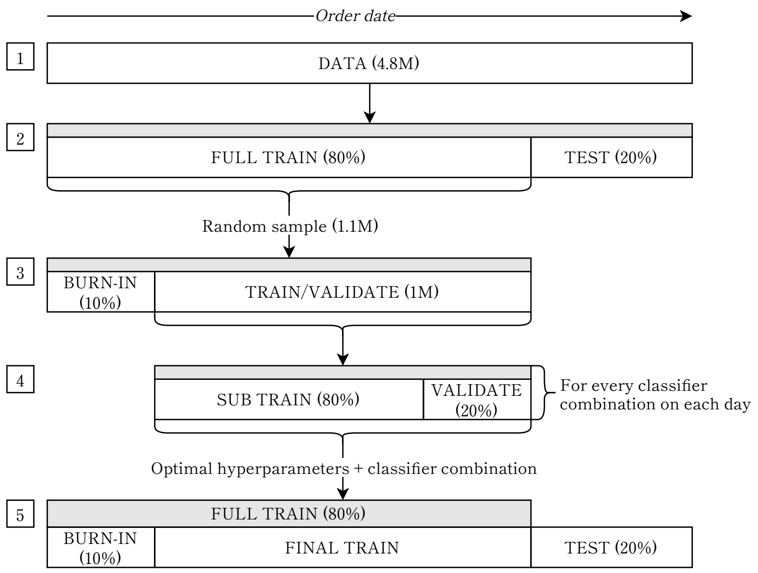


Fig. 5. Procedure for training, validation, and testing. Between brackets, we indicate the number of observations or the percentage of a certain set of observations. M means 'million'.

and the corresponding optimal hyperparameters. To do so, we divide our training data into a training set and a validation set. For each combination of classifiers, we then turn to Bayesian hyperparameter optimisation (Tree-structured Parzen Estimation) [2, 18], the implementation of which is elaborated on in Section A. Training our hierarchy using LCPN, we evaluate the validation performance by means of top-down mandatory leaf-node prediction, where we aim to maximise the validation hF_1, which is defined in (6). Having obtained a set of optimal hyperparameters for all eight classifier combinations, we compare the hF_1-scores for all these combinations and select the one with the best performance. Applying this procedure, we obtain daily optimal hierarchies.

To illustrate the classifier selection and optimisation procedure, we visualise our train-validation splits in Figure 5. First, we split the data into train (80%) and test (20%) using a time series split, so that all test data comes after train data in terms of order date (Step 1 and 2), which is most representative of the real-life situation. Next, we take a subsample of the train data containing 1.1 million instances. As some of our engineered features. i.e., our *historic* variables, require some time to stabilise, we decide to burn in the first 10% of observations, leaving us with a sample of 1 million observations to train and validate on (Step 3). After that, we split this sample into train (80%) and validation (20%), again using a time series split (Step 4). On this split, we apply the previously described procedure of finding the best daily combination of classifiers with optimal hyperparameters to this train-validation split to find optimal daily hierarchies.

4.5. Certainty-Based Automated Thresholds (CAT)

In this section, we present the CAT algorithm, which is a blocking approach for handling NMLNP. In a given hierarchy, CAT independently computes thresholds for all child nodes $m \in \mathcal{S}$. When dealing with a time component (in HCOT), we have daily hierarchical classifiers, such that thresholds will be computed for each child node and each day t independently. The threshold decides whether an instance coming from the parent node of m, p_m, gets accepted at node m. We denote this threshold by $\text{Th}_t(m)$. Since our hierarchy consists of seven child nodes and HCOT has 10 days of non-mandatory leaf-node prediction, applying CAT to HCOT implies computing 10 (days) $\times$ 7 (nodes per day) = 70 thresholds. For a single threshold in $\mathcal{H}_t$, we first obtain the class probabilities of the trained instances based on true data, i.e., only the instances truly belonging to p_m are considered for the threshold of node m, and then use the corresponding set of probabilities to compute the optimal threshold.

First, we introduce the crux of the CAT algorithm. This is the constant hyperparameter α, which we dub the 'certainty' parameter. The hyperparameter is used to locate a lower bound for the optimisation of the thresholds. The certainty α decides the trade-off between accuracy and timeliness. Since there is no mathematical optimum in this trade-off, the certainty represents user preferences where accuracy might be more or less important than timeliness. Hence, the hyperparameter α is user-specified.

Second, we introduce some terminology to formally define the CAT algorithm and context. We denote the set of instances that, during the training phase, is known to belong to parent node p_m as $\text{Training}(p_m)$. The posterior probability that train instance $u_j \in \text{Training}(p_m)$ belongs to node m, as assigned by the classifier at parent node p_m on day t, is denoted as $\gamma_{p_m \to m,t}(u_j)$. Next, we define the set with majority vote probabilities, which are the probabilities for node m that exceed the probabilities to go to any one of the sibling nodes of m, $m' \in \mathcal{S}_m$. Formally, this is given by

$$\mathcal{Z} = \left\{ \gamma_{p_m \to m,t}(u_j) \;\middle|\; \gamma_{p_m \to m,t}(u_j) > \gamma_{p_m \to m',t}(u_j), \quad \forall m' \neq m,\, m' \in \mathcal{S}_m, \quad \forall u_j \in \text{Training}(p_m) \right\}. \tag{1}$$

Since instances can only belong to one class in this problem setting, they are assigned to exactly one node. Logically, node m is only presented with majority vote probabilities, where the threshold subsequently decides whether an instance is accepted. Therefore, we train $\text{Th}_t(m)$ solely on instances that have a majority vote probability for node m on day t.

Similarly to $\text{Training}(p_m)$, we define $\text{Training}(m)$ and $\text{Training}(\mathcal{S}_m)$. The threshold should block instances that do not belong to $\text{Training}(m)$ and should accept instances that do. To distinguish between instances that should and should not be accepted, we first define the list of majority vote probabilities that should be accepted by the threshold as

$$\zeta_t(m) = \left\{ \gamma_{p_m \to m,t}(u_j) \in \mathcal{Z} \;\middle|\; u_j \in \text{Training}(m) \right\}. \tag{2}$$

The list of majority vote probabilities that should be accepted to one of the sibling nodes of m, ergo, not to node m, is given by

$$\zeta_t(\neg m) = \left\{ \gamma_{p_m \to m,t}(u_j) \in \mathcal{Z} \;\middle|\; u_j \in \text{Training}(\mathcal{S}_m) \right\}. \tag{3}$$

Now, the certainty parameter α comes in and its role is quite straightforward. The certainty is a number between 0 and 1 and it is used to compute the (100α)-th percentile of $\zeta_t(\neg m)$, denoted by $\zeta_t(\neg m)_\alpha$. The percentile serves as a lower bound on the search space for the optimal threshold. This search space is dubbed the 'allowed search space' $\mathcal{V}$, which is formally defined as

$$\mathcal{V} = \left\{ \gamma_{p_m \to m,t}(u_j) \in \mathcal{Z} \;\middle|\; \gamma_{p_m \to m,t}(u_j) \geq \zeta_t(\neg m)_\alpha \right\} . \tag{4}$$

Essentially, CAT allows us to be certain that *at least* $(100\alpha)\%$ of $\zeta_t(\neg m)$ will be blocked by $\text{Th}_t(m)$. When $\alpha = 0$, no restrictions are placed on the allowed search space. This is likely to yield relatively low thresholds and result in timely, but relatively less accurate classifications. On the other hand, when $\alpha = 1$, we restrict $\mathcal{V}$ to contain probabilities that are only found in $\zeta_t(m)$, which is likely to result in thresholds approaching 1. Consequently, this will yield highly accurate but less timely classifications. Also, note that the relation between α and the relative importance of timeliness and accuracy is not necessarily linear but dependent on the data set.

Subsequently, within $\mathcal{V}$, we optimise threshold $\text{Th}_t(m)$ by looking for the separation point between $\zeta_t(m)$ and $\zeta_t(\neg m)$ that maximises F_1. To do so, CAT requires a classifier that outputs near-continuous probabilities. Otherwise, the instances will be concentrated in the few available probabilities, such that making a distinctive split is impossible. Given a near-continuous classifier, it is not necessary to consider each probability as potential threshold, as the number of distinct probabilities is often extremely high. Hence, we do a grid search using ξ evenly spaced steps within $\mathcal{V}$. The hyperparameter ξ is given as input to CAT. We denote the set containing the ξ grid points as Θ, where each $\theta \in \Theta$ is considered as potential threshold.

For each potential threshold, we determine the number of true positives (TP) and false positives (FP) by calculating the number of posterior probabilities in $\zeta_t(m)$ and $\zeta_t(\neg m)$ that exceed the threshold, respectively. Next, we calculate precision and recall, where the latter uses that the sum of true positives (TP) and false negatives (FN) equals the cardinality of $\zeta_t(m)$. Combining precision and recall, we compute the F_1-measure and select the threshold $\text{Th}_t(m)$ with the highest F_1. Algorithm 1 presents the complete procedure of CAT.

When applying CAT to HCOT, we deal with a time component. Over time, our classifier is based on more information, making it easier to separate probabilities. This means that the posterior probabilities gravitate towards 0 and 1. If a probability moves towards 1, it is more likely to stay or become a majority vote probability. The other way around, if it moves towards 0, it becomes less likely that the probability will remain a majority vote probability. Consequently, the distributions of $\zeta_t(m)$ and $\zeta_t(\neg m)$ are more likely to shift towards 1 over time because they only contain majority vote probabilities. In general, this will increase the lower bound of $\mathcal{V}$ and hence, may push $\text{Th}_t(m)$ upward. Whether the threshold increases over time depends on the distributions of $\zeta_t(m)$ and $\zeta_t(\neg m)$, as well as on α.

Figure 6 visualises the working of the CAT algorithm, showing example distributions of the majority vote probability sets $\zeta_t(m)$ and $\zeta_t(\neg m)$. A probability in $\zeta_t(m)$ should be accepted, because the instance belongs to class m. In contrast, a probability in $\zeta_t(\neg m)$ should not be accepted, because the instance does not belong to m. The allowed search space $\mathcal{V}$ is represented by the shaded region. A small circle shows the F_1-measure evaluated in a point in the grid. The optimal threshold is presented by the bigger circle.

Algorithm 1: CAT

```
   Data: $\{\gamma_{p_m\to m,t}(u_j) \mid u_j \in \text{Training}(p_m)\};\ \alpha;\ \xi$
   Result: $\text{Th}_t(m)$
1  Initialise the following sets
       $\mathcal{Z} = \{\gamma_{p_m\to m,t}(u_j) \mid \gamma_{p_m\to m,t}(u_j) > \gamma_{p_m\to m',t}(u_j),\quad \forall m' \neq m,\ m' \in \mathcal{S}_m,\ \forall u_j\}$
       $\zeta_t(m) = \{\gamma_{p_m\to m,t}(u_j) \in \mathcal{Z} \mid u_j \in \text{Training}(m)\}$
       $\zeta_t(\neg m) = \{\gamma_{p_m\to m,t}(u_j) \in \mathcal{Z} \mid u_j \in \text{Training}(\mathcal{S}_m)\}$
2  Define the allowed search space, $\mathcal{V} = \{\gamma_{p_m\to m,t}(u_j) \in \mathcal{Z} \mid \gamma_{p_m\to m,t}(u_j) \geq \zeta_t(\neg m)_\alpha\}$
3  Define the set with potential thresholds $\Theta$ as $\xi$ evenly spaced points within $[\min\{\mathcal{V}\}, \max\{\mathcal{V}\}]$
4  Initialise $F_1^* = 0$ and $\text{Th}_t(m) = 0$
5  for $\theta \in \Theta$ do
6      $\text{TP} = |\{\gamma_{p_m\to m,t}(u_j) \in \zeta_t(m) \mid \gamma_{p_m\to m,t}(u_j) \geq \theta\}|$
7      $\text{FP} = |\{\gamma_{p_m\to m,t}(u_j) \in \zeta_t(\neg m) \mid \gamma_{p_m\to m,t}(u_j) \geq \theta\}|$
8      $recall = \frac{\text{TP}}{|\zeta_t(m)|};\quad precision = \frac{\text{TP}}{\text{TP}+\text{FP}};\quad F_1 = \frac{2 \times precision \times recall}{precision + recall}$
9      if $F_1 > F_1^*$ then
10         $F_1^* := F_1$
11         $\text{Th}_t(m) := \theta$
```

In this example, the effect of α is straightforward. Namely, if α increases, the lower bound of the allowed search space moves to the right. If it moves far enough, the optimal threshold $\text{Th}_t(m)$ will be pushed upwards. This will lead to less instances from $\zeta_t(\neg m)$ being accepted at node m, which increases accuracy. However, since less instances are classified, we obtain less timely predictions. The other way around, if α decreases, the allowed search space widens, which may result in a lower optimal threshold. If this is the case, more instances are accepted, which leads to more timely yet less accurate predictions. Note that in this example, $\text{Th}_t(m)$ would not decrease with a lower α.

With respect to HCOT, predicting a false negative is not as troublesome as predicting a false positive. Namely, if an instance is blocked, we can try again the next day, but if an instance is accepted and thereby classified to the wrong leaf node, there is no chance to rectify this mistake. While the predictions should be accurate, they should also be timely. Therefore, we solve this issue with CAT, where we use α to reach a certain standard for precision, but at the same time try to recall as many instances as possible within the allowed search space. Hence, we optimise F_1 to give equal weight to precision and recall within $\mathcal{V}$.

4.6. CAT-HCOT

Having obtained the daily best classifier combination and respective optimal hyperparameters as well as a procedure to compute daily optimal thresholds, we can now predict the quality of orders by combining CAT and HCOT, in short denoted by CAT-HCOT. First, we train the complete HCOT algorithm by independently training each $\mathcal{H}_t$, for $t = 0, 1, \dots, 10$, with the obtained optimal hyperparameters and best classifier

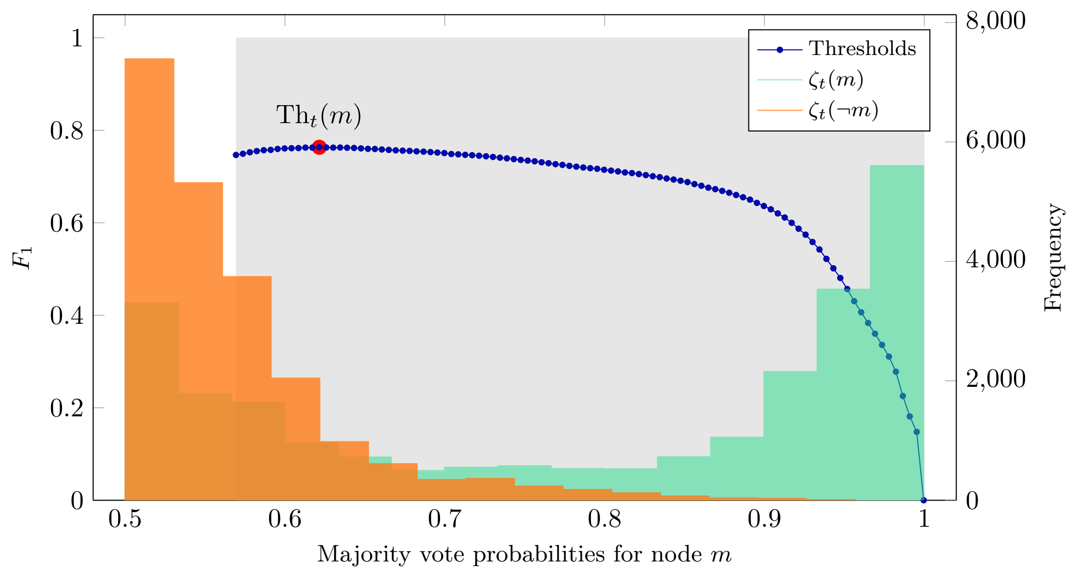


Fig. 6. Visualisation of example distributions of $\zeta_t(m)$ and $\zeta_t(\neg m)$ alongside the optimisation of F_1 within $\mathcal{V}$.

combinations. We do this using the previously explained LCPN approach. Note that for each hierarchical classification we thus train on the same (number of) instances and the same features, while the information contained in those features may change over time.

Second, we apply CAT with certainty parameter α independently to each $\mathcal{H}_t$ to obtain day- and node-specific thresholds $\text{Th}_t(m)$. Similar to the trained classifiers, the thresholds are computed based on the true training labels and their respective class probabilities. That is, only instances truly belonging to a specific parent node are considered when computing the thresholds belonging to its child nodes.

Third, combining the trained hierarchical classifiers and their optimal thresholds, we can start predicting instances in a top-down, non-mandatory leaf-node procedure over time. Algorithm 2 presents this procedure. In short, we only classify instances when we are confident enough about their classification and otherwise try again the next day when (possibly) new determinants or other information is available. This leads to some instances being classified sooner than other instances, with the overarching goal of balancing accuracy and timeliness.

In applying these three steps, we consider the full training set containing 80% of data points and burn in its first 10% of instances (sorted by order date) to stabilise our historic variables (Figure 5, Step 5). The withheld test set containing the most recent 20% of data is then run through Algorithm 2 in order to evaluate the overall performance of CAT-HCOT.

4.7. Performance Measures

To evaluate the performance of CAT-HCOT, we use the measure of predictive accuracy alongside hierarchical performance measures as proposed by Kiritchenko et al. [17]. These latter measures are the hierarchical extensions of precision, recall, and F_1-score. Hierarchical performance measures explicitly incorporate the

```
Algorithm 2: CAT-HCOT
Data: Set $\mathcal{R}$ of all test instances $r_i = (\mathbf{x}_i, l_i)$ to be classified; Tree-based class hierarchy $\mathcal{H}$; Trained
      hierarchy $\mathcal{H}_t$ at $t \in \{0, 1, \dots, 10\}$; Thresholds $\text{Th}_t(m)$ for node $m$ at day $t$ from CAT
Result: Set $\mathcal{P}$ of classified test instances $(\mathbf{x}_i, \hat{l}_i)$, where $\hat{l}_i \in \mathcal{L}$
1  Initialise $\mathcal{P} = \emptyset$ and $t = 0$
2  while $\mathcal{R} \neq \emptyset$ do
3      for $r_i \in \mathcal{R}$ do
4          if $t < 10$ then
5              Top-down, non-mandatory leaf-node prediction with respect to $\mathcal{H}_t$ assigns $\mathbf{x}_i$ a label
                $\hat{l}_i \in \mathcal{I} \cup \mathcal{L}$ if posterior probabilities exceed thresholds $\text{Th}_t(m)$ for the entire path to $\hat{l}_i$
6              if $\hat{l}_i \in \mathcal{L}$ then
7                  $\mathcal{R} := \mathcal{R} \setminus r_i$
8                  $\mathcal{P} := \mathcal{P} \cup (\mathbf{x}_i, \hat{l}_i)$
9          else
10             Top-down, mandatory leaf-node prediction with respect to $\mathcal{H}_{10}$ assigns $\mathbf{x}_i$ a label $\hat{l}_i \in \mathcal{L}$
11             $\mathcal{R} := \mathcal{R} \setminus r_i$
12             $\mathcal{P} := \mathcal{P} \cup (\mathbf{x}_i, \hat{l}_i)$
13     $t := t + 1$
```

hierarchical path-distance between true and predicted labels, which means that the farther classifications are from the true label with respect to the hierarchy, the heavier the penalty. This property makes hierarchical performance measures preferable over flat performance measures in case of hierarchical classification.

Global predictive accuracy (Acc), *global hierarchical precision* (hP) and *global hierarchical recall* (hR) are defined as follows:

$$Acc = \frac{1}{N}\sum_{i=1}^{N}\left|\mathcal{A}(l_i) \cap \mathcal{A}(\hat{l}_i)\right|, \qquad hP = \frac{\sum_{i=1}^{N}\left|\mathcal{A}(l_i) \cap \mathcal{A}(\hat{l}_i)\right|}{\sum_{i=1}^{N}\left|\mathcal{A}(\hat{l}_i)\right|}, \qquad hR = \frac{\sum_{i=1}^{N}\left|\mathcal{A}(l_i) \cap \mathcal{A}(\hat{l}_i)\right|}{\sum_{i=1}^{N}\left|\mathcal{A}(l_i)\right|}, \tag{5}$$

where l_i and $\hat{l}_i$ denote the true and the predicted label of instance i, respectively, and $\mathcal{A}(l)$ denotes the set containing leaf node $l \in \mathcal{L}$ and all its ancestor nodes except the root node R. We use the term 'global' here, because these performance measures are calculated over all N test instances after the total period of 10 days.

The *global hierarchical F_1-measure* (hF_1) aggregates the hP and hR measures by taking their harmonic mean, and is defined as follows:

$$hF_1 = \frac{2 \cdot hP \cdot hR}{hP + hR}. \tag{6}$$

In the case of flat precision and recall, we can compute class-specific precision and recall. We extend this idea to our hierarchical measures by defining *class-specific hierarchical precision* (hP_c) and *class-specific* *hierarchical recall* (hR_c), except that we now sum over all predicted instances of class $c \in \mathcal{L}$ in hP_c, and all true instances of class $c \in \mathcal{L}$ in hR_c. We calculate the *class-specific hierarchical $F_{1,c}$-measure* ($hF_{1,c}$), where we replace hP and hR by their class-specific counterparts hP_c and hR_c.

As our HCOT algorithm classifies instances over time, we also want to evaluate its performance per day. This means that we need daily hierarchical performance measures. Therefore, we define *daily accuracy* (Acc_t), *daily hierarchical precision* (hP_t), and *daily hierarchical recall* (hR_t), where we now sum over (and divide by) all N_t instances that are assigned a (leaf) class $c \in \mathcal{L}$ on day $t \in \{0, 1, \ldots, 10\}$. We calculate the *daily hierarchical $F_{1,t}$-measure* ($hF_{1,t}$), replacing hP and hR by their daily counterparts hP_t and hR_t.

Last, we combine the class-specific and daily hierarchical performance measures to calculate *class-specific daily hierarchical precision* ($hP_{c,t}$) and *class-specific daily hierarchical recall* ($hR_{c,t}$). As before, these measures can be combined to calculate the *class-specific daily hierarchical $F_{1,c,t}$-measure* ($hF_{1,c,t}$).

To evaluate the extent to which instances are blocked at a certain classifier on a parent node, we use the *blocking factor*, which is defined as the fraction of instances that is blocked at a parent node relative to the total number of instances entering the corresponding classifier in the top-down class-prediction approach. The blocking factor gives us insight into the evolution of the blocking process at each parent node over time.

4.8. Baseline Methods

We compare the performance of CAT-HCOT against two baselines: static HCOT and flat CAT-HCOT.

4.8.1. Static HCOT

We first compare the performance of CAT-HCOT against *static HCOT*, which is a hierarchical classification method with MLNP. More specifically, this baseline method does not use a blocking approach with thresholds, such that, each day, it classifies all instances. When using static HCOT in practice, one would choose a specific day and use the predictions obtained for that day. Considering the accuracy-timeliness trade-off, we expect CAT-HCOT to achieve higher daily accuracy than static HCOT for the first few days, since it only assigns instances to leaf nodes if there is enough confidence to do so. For the last few days, we expect static HCOT to obtain higher daily accuracies, since, at this point, CAT-HCOT would only need to predict previously blocked and thus more uncertain instances. In terms of timeliness, static HCOT will become relatively less timely across days, as CAT-HCOT classifies more and more instances over time.

4.8.2. Flat CAT-HCOT

We also compare the performance of CAT-HCOT against *flat CAT-HCOT*, which is a specific case of CAT-HCOT with the simplest hierarchy possible: one root node that has all leaf classes as its child nodes. This flat baseline method uses the same classifier (with $C = 2^1 = 2$ combinations) and model optimisation approach, and applies the same (CAT) blocking approach as standard CAT-HCOT. As for the accuracy-timeliness trade-off, we expect the accuracy for flat CAT-HCOT to be slightly lower than that of CAT-HCOT, because the smaller hierarchy is likely to capture the hierarchical structure in the classes less accurately. Presumably, this would most affect the least represented classes in the data (i.e., Medium and Heavily Deficient). Note that, even though the flat baseline uses a one-level hierarchy, we use the same set of ancestor nodes as in CAT-HCOT when evaluating hierarchical performance measures (such that, e.g., a Medium Deficient Product Order is also Conclusive and Deficient). In terms of timeliness, we expect flat CAT-HCOT to provide slightly more timely predictions than regular CAT-HCOT, since its one-level hierarchy causes instances to be classified already when a single threshold is passed.

4.9. Software

We use Python 3.8 for implementation. To program the LR and RF classifiers, we use the *scikit-learn* package [20]. We perform hyperparameter optimisation using the *hyperopt* package [3]. To program the hierarchical structure, we use the framework as presented by Warshaw [27]. Our code is available on GitHub at https://github.com/thomas-brink/CAT-HCOT.

5. Results

In this section, we present and evaluate our results. First, we discuss the found best classifier combinations and optimal model hyperparameters. Second, we consider the optimal thresholds as determined by our CAT algorithm. Third, we evaluate the predictive performance of CAT-HCOT and, fourth, compare it against two baseline methods. Last, we perform a sensitivity analysis on the certainty parameter α.

5.1. Classifier Selection and Optimisation

As previously described, we select and optimise our classifiers by performing an exhaustive search across all classifier combinations at the parent nodes in our hierarchy. For CAT-HCOT, we find that a hierarchy with LR on node <R> and RF on nodes <1> and <1.2> is the dominant choice. Namely, across all 8 combinations, LR-RF-RF yields the highest validation F1-score for 10 out of 11 daily hierarchical classifiers. Therefore, to make implementation and interpretation simple, we decide to work with this classifier combination for each of the eleven daily hierarchical classifiers. This combination confirms our notion that different classifiers may be better at distinguishing between different 'regions' of the data, which is an important factor why hierarchical classification may be preferred over flat classification. For flat CAT-HCOT, we find that LR is the optimal choice for $\mathcal{H}_0, ..., \mathcal{H}_4$, while RF yields the highest validation F_1-score from $\mathcal{H}_5$ onwards. Table A.1 presents the optimised hyperparameters for CAT-HCOT (LR-RF-RF) and flat CAT-HCOT (LR and RF) for all hierarchical classifiers $\mathcal{H}_0, ..., \mathcal{H}_{10}$.

5.2. Certainty-Based Automated Thresholds (CAT)

Next, we discuss the optimal thresholds as determined by the CAT algorithm. We set the certainty parameter $\alpha = 0.7$ and use $\xi = 100$ number of steps in the allowed search space. We point out that these choices are user-specific and thus remain somewhat arbitrary. Although no 'optimal' value of α exists,

Table 2
Optimal thresholds per node and hierarchy computed with the CAT algorithm where $\alpha = 0.7$ and $\xi = 100$.

	$\mathcal{H}_0$	$\mathcal{H}_1$	$\mathcal{H}_2$	$\mathcal{H}_3$	$\mathcal{H}_4$	$\mathcal{H}_5$	$\mathcal{H}_6$	$\mathcal{H}_7$	$\mathcal{H}_8$	$\mathcal{H}_9$
Inconclusive	95.8%	96.4%	96.5%	96.8%	96.6%	97.5%	98.1%	98.7%	99.0%	99.2%
Conclusive	89.8%	89.3%	80.2%	73.2%	66.5%	64.2%	66.9%	88.2%	83.2%	77.5%
Perfect	64.1%	66.1%	74.9%	78.0%	80.8%	84.0%	82.8%	82.9%	84.8%	85.8%
Deficient	63.4%	59.2%	63.5%	63.5%	69.7%	68.0%	71.9%	71.8%	71.3%	71.7%
Mildly Deficient	57.0%	58.3%	59.8%	62.0%	64.3%	66.3%	69.1%	70.4%	70.9%	71.0%
Medium Deficient	51.9%	52.2%	52.4%	52.9%	53.5%	53.9%	55.6%	57.4%	59.6%	61.1%
Heavily Deficient	50.3%	48.8%	48.2%	47.5%	47.9%	47.1%	47.6%	47.6%	48.1%	49.7%

$\alpha = 0.7$ is an appropriate choice, putting a relatively strong weight on accuracy yet still ensuring timely predictions. Section 5.5 provides further insight into the dynamics behind different values of α.

Table 2 shows the optimal thresholds for each daily hierarchy per node as determined by CAT. In general, we would expect the thresholds to slightly increase over time. This is the case for almost all nodes, apart from some minor fluctuations. Notable is the dip around $\mathcal{H}_5$ for node <1> (Conclusive), which means there is some variability in the distributions of $\zeta_t(<1>)$ and $\zeta_t(\neg <1>)$. Additionally, we note that the thresholds tend to be higher for nodes at the top of the hierarchy (e.g., Inconclusive and Conclusive) compared to nodes deeper in the hierarchy (e.g., Mildly, Medium, and Heavily Deficient). A high threshold likely results from a high lower bound, which means the classifier had difficulty separating the distributions of $\zeta_t(m)$ and $\zeta_t(\neg m)$. We point out that one should be careful in comparing thresholds for the Deficient classes with the other classes, since the range of majority vote probabilities of the Deficient classes differs from that of the other classes. This is due to the multi-class split that the classifier at node <1.2> (Deficient) makes, as opposed to the binary splits at the other two parent nodes.

5.3. *CAT-HCOT*

Employing the best classifier combination with optimal hyperparameters and using the optimal thresholds from CAT ($\alpha = 0.7$), we use the CAT-HCOT algorithm to classify our test instances. In this section, we evaluate the predictive performance of CAT-HCOT. We start by discussing its global and class-specific performance, after which we evaluate the use of the blocking approach.

5.3.1. *General Performance*

Overall, we find that CAT-HCOT achieves a global accuracy of 93.7%. Furthermore, Table 3 shows global and class-specific hierarchical performance measures. All performance measures in Table 3 are computed based on the entire set of leaf-node predictions that is made in the entire period of 0 to 10 days after the order date. Here, global measures are computed based on all classified instances, whereas class-specific measures are computed based on all instances that are assigned to a specific class. Global hierarchical precision, recall, and F_1-scores all reach values over 90%. If we consider class-specific measures, the scores remain high with values mostly above 90%. One particular class that deviates somewhat is Inconclusive, which achieves a

Table 3
Global and class-specific performance of CAT-HCOT.

	hP	hR	hF_1
Global	96.1%	92.2%	94.1%
Perfect	96.8%	94.3%	95.6%
Inconclusive	93.6%	83.0%	87.9%
Mildly Deficient	97.2%	97.7%	97.4%
Medium Deficient	92.7%	92.9%	92.8%
Heavily Deficient	95.2%	95.4%	95.3%

relatively low hierarchical recall score of 83%. This finding could be explained by the fact that instances only need to pass a single threshold to be classified as Inconclusive. Hence, for each instance, hierarchical recall is either 0 or 1, whereas for other classes, it can achieve scores between 0 and 1. As a result, hierarchical recall (and thus hierarchical F_1) could be lower for Inconclusive than for other classes.

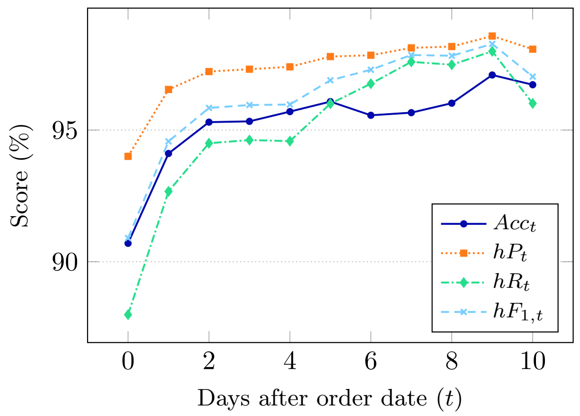
Figure 7 presents the performance of the CAT-HCOT algorithm in terms of daily accuracy (Acc_t), daily hierarchical precision (hP_t), daily hierarchical recall (hR_t), daily hierarchical F_1-score ($hF_{1,t}$), and the cumulative percentage of classified instances per day. The daily hierarchical performance measures evaluate all leaf-node predictions that are made on a specific day. Considering daily accuracy (Figure 7a) and the cumulative percentage of leaf-node predictions (Figure 7b), we find that CAT-HCOT is able to classify roughly 40% of instances on the order date itself with an accuracy of almost 91%. As more determinants come available over time, the cumulative percentage of leaf classifications increases further (with decreasing rate). On the 5th day after ordering, CAT-HCOT has leaf-classified almost 80% of instances, where the level seems to stabilise (Figure 7b). After day 5, the cumulative percentage of leaf classifications almost stops increasing, which means that roughly 20% of instances cannot be classified with enough certainty. Hence, on day 10, we make sure all these instances are classified by means of MLNP, as noticeable by the peak on day 10. We point out that from day 2 onwards (including day 10), CAT-HCOT achieves a daily accuracy of 95% or higher on the daily leaf-classified instances (Figure 7a).

For instances that are blocked at a particular parent node and therefore not assigned a leaf class, CAT-HCOT may be able to assign them a predicted class by classifying them into one of the internal nodes Conclusive or Deficient. It is interesting to evaluate the cumulative percentage of internal-node predictions (excluding the root node), because this gives a better indication of the practical use of the algorithm (upper line in Figure 7b). That is, e-commerce platforms do not necessarily need detailed labels to act upon predictions, but can in some cases also rely on more general predictions. For example, if a product order is predicted to be Deficient, this already may be valuable enough for the company to act upon. From Figure 7b, we find that roughly 64% of instances are assigned an internal- or leaf-node label on the order date itself, which further increases towards roughly 93% on day 9. Here, we recall that the internal-node predictions are never finalised. Instead, the associated instances are again classified in a top-down manner on the next day, until they reach a leaf node.

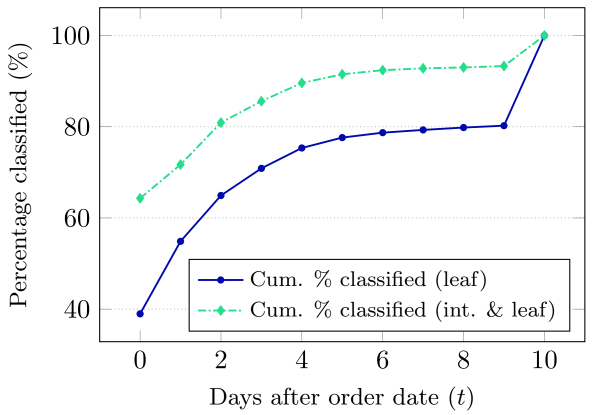
Last, we evaluate the performance of CAT-HCOT on a daily, class-specific basis. Figure 8 presents daily class-specific hierarchical precision ($hP_{c,t}$) and recall ($hR_{c,t}$) evaluated for all five leaf classes. We find that almost all daily class-specific hierarchical scores exceed 90%, which indicates that CAT-HCOT achieves very strong predictive performance for all classes. There are two exceptions to this result. First, hierarchical recall for Inconclusive shows a deviating pattern over time: it starts relatively low, just below 80%, and increases slowly over time, eventually exceeding 90% (Figure 8b). Again, this can be explained by the fact that Inconclusive is positioned on top of the hierarchy, such that instances only need to pass one threshold to be classified as Inconclusive. As a result, for each instance, hierarchical recall is either 0 or 1, leading to lower scores on the first days when clearly not all true Inconclusive instances have been correctly predicted. The second exception is related to the classes Mildly Deficient on day 0 and Heavily Deficient on day 10: in both cases, hierarchical precision and recall achieve values below 80%. A possible explanation for this is the low occurrence of these classes and availability of determinants in the data, which makes accurate classification more difficult, even if we impose certainty thresholds and allow for classification over time.

5.3.2. Blocking at Parent Nodes

In this section, we give more insight into the CAT blocking approach, which is used to stop classification of instances if the posterior probability at a given level does not exceed a threshold. Figure 9a shows the daily blocking factor for the three different parent nodes in our hierarchy. Over time, we find that the blocking factor decreases for the root node <R>, while it increases for parent nodes <1> and <1.2>. This can be explained by inspecting Figure 9b, where we plot the percentage of blocked instances relative to the (fixed) total number of instances that need to be classified. Over time, this percentage stays relatively constant for nodes <1> and <1.2>, while the daily total fraction of blocked instances, given by the sum of the percentages at the three parent nodes, decreases over time. Thus, in absolute terms, the number of

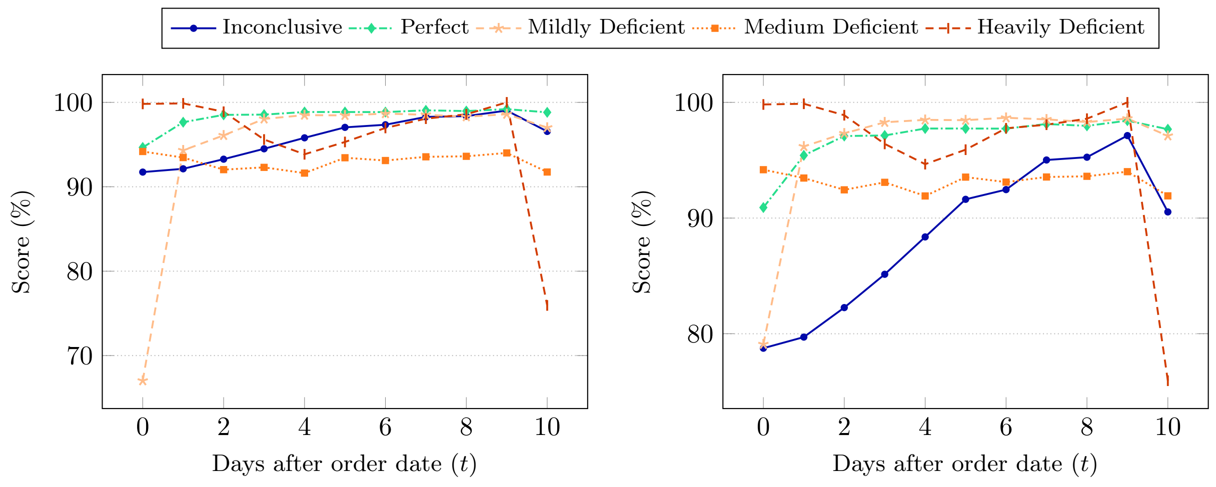


(a) Daily hierarchical performance measures.



(b) Cumulative percentage of classified instances for internal- and leaf-node predictions (excluding the root node).

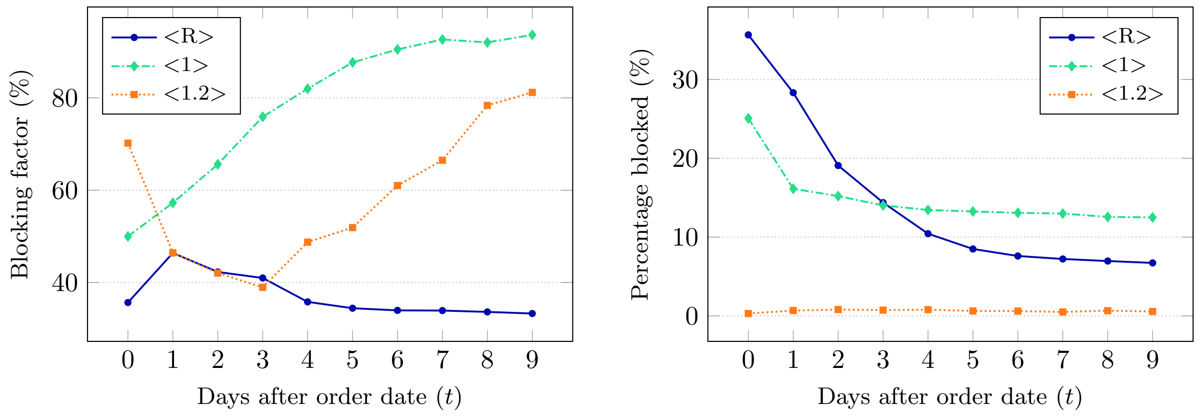
Fig. 7. Daily performance of the CAT-HCOT algorithm.



(a) Daily class-specific hierarchical precision ($hP_{c,t}$).

(b) Daily class-specific hierarchical recall ($hR_{c,t}$).

Fig. 8. Daily class-specific hierarchical performance measures.



(a) Daily blocking factor per parent node.

(b) Blocked instances as a percentage of all instances.

Fig. 9. Analysis of the blocking procedure at the parent nodes.

blocked instances at nodes <1> and <1.2> is rather stable, but the number of instances entering these nodes decreases, such that the blocking factor increases. Note that up until day 9, at least 6.7% of instances are blocked at the root node <R> (Figure 9b). These instances are not assigned an internal (excluding root) classification and therefore do not provide us with actionable business insights. Logically, since we apply MLNP on day 10, the fraction of blocked instances in the end reaches 0 for all parent nodes.

Even if there is not enough certainty of providing leaf-node predictions, CAT-HCOT still provides actionable insights by assigning internal labels. We do not finalise these internal-node predictions, meaning that the next day the instances are again classified in a top-down manner and may end up in a different (internal) node. This procedure helps us overcome the issue of error propagation. Figure 10 shows the performance at the two internal nodes <1> (Conclusive) and <1.2> (Deficient). First, we observe that hierarchical recall is relatively low for both nodes. This is not surprising, since, by definition, internal-node predictions have not yet reached the most specific node possible, making it impossible to achieve perfect recall scores. More

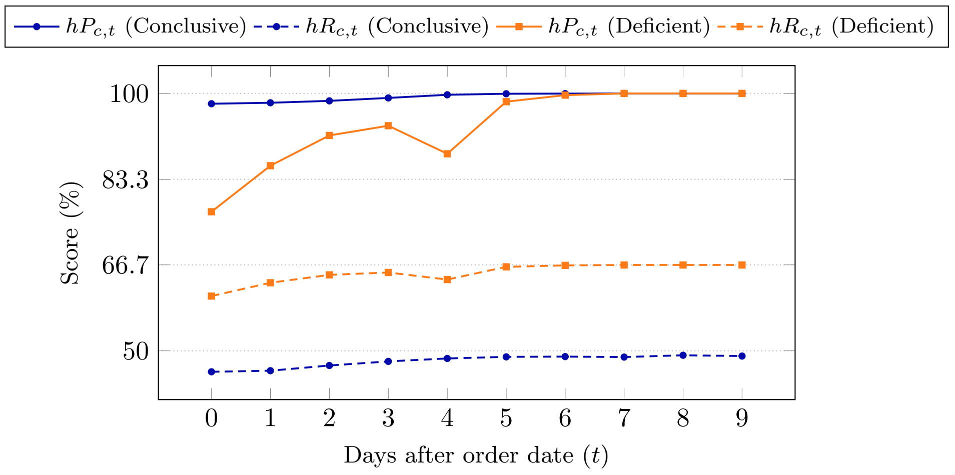


Fig. 10. Daily internal-node hierarchical performance measures (excluding root). Daily class-specific hierarchical recall ($hR_{c,t}$) can reach maximum values of 50% and 66.7% for <1> (Conclusive) and <1.2> (Deficient), respectively.

specifically, daily class-specific hierarchical recall ($hR_{c,t}$) can reach maxima of 50% and 66.7% for <1> and <1.2>, respectively, since at most 1 out of 2 and 2 out of 3 nodes in the path can be assigned correctly. Second, we find that hierarchical precision is close to 100% on all days for <1>, while predictions are almost always perfectly precise for <1.2> from day 5 onwards. Still, a significant fraction of internal predictions is incorrect for <1.2> on day 0 until day 4. This shows that, indeed, it may not be a good idea to finalise internal predictions, as this would propagate the error to the next day by continuing classification from an incorrect node. However, as the internal-node predictions are nearly always correct after day 4, it could be beneficial to finalise the internal classifications from this day onwards.

5.4. Baseline Methods

In this section, we evaluate the performance of CAT-HCOT against two baselines: static HCOT, which does not use a blocking approach, and flat CAT-HCOT, which uses a one-level hierarchy. Figure 11 plots daily accuracy scores and cumulative percentages of classified instances for CAT-HCOT and the two baselines.

We first compare CAT-HCOT to the static baseline. For static HCOT, we achieve an accuracy of 81.7% when predicting all instances on day 0 (Figure 11a). This accuracy steadily increases until reaching a value of 98.2% on day 10. This increase makes sense, as with each day more determinants become available, which in turn leads to more accurate predictions. In contrast to static HCOT, CAT-HCOT does not assign leaf-node predictions for all instances at each day, but only classifies those instances for which it is certain enough. This leads to higher daily accuracy than static HCOT for the first 5 days after (and including) the order date. In contrast, from day 6 to day 10, static HCOT attains higher daily accuracy than CAT-HCOT. This is mainly because in the later days, CAT-HCOT only needs to classify those instances that were previously blocked and are thus difficult to classify to a leaf node, whereas static HCOT classifies the full set of instances each day, as seen by the straight line in Figure 11b. All in all, we find that CAT-HCOT adequately balances the accuracy vs. timeliness trade-off by using a certainty-based classification approach, while static HCOT

Table 4
Global and class-specific hierarchical performance of the flat CAT-HCOT baseline ($\alpha = 0.7$).

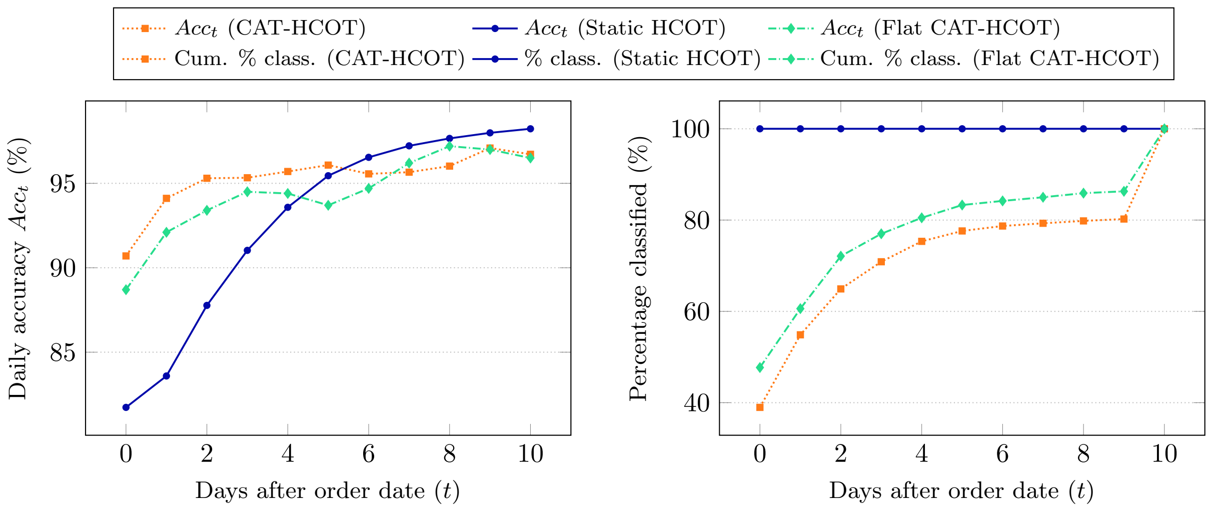
	hP	hR	hF_1
Global	93.7%	90.9%	92.2%
Perfect	96.3%	93.4%	94.8%
Inconclusive	93.9%	83.9%	88.6%
Mildly Deficient	89.2%	93.7%	91.4%
Medium Deficient	63.8%	77.7%	70.1%
Heavily Deficient	55.5%	77.9%	64.8%

does not allow for such a balanced trade-off.

Second, we compare CAT-HCOT to the flat baseline ($\alpha = 0.7$). From Figure 11a, we find that CAT-HCOT achieves higher accuracy scores than flat CAT-HCOT, while it provides less timely predictions. However, these differences are small. Instead, CAT-HCOT and flat CAT-HCOT mostly differ in terms of class-specific hierarchical performance. In particular, we find that flat CAT-HCOT is not able to achieve similarly high hierarchical precision and recall scores as CAT-HCOT on the Deficient classes. While CAT-HCOT achieves hierarchical precision and recall scores of above 90% for Medium and Heavily Deficient (Table 3), flat CAT-HCOT only achieves scores between 50% and 80% (Table 4). Hence, taking into account the full hierarchical class structure, CAT-HCOT achieves major improvements in class-specific predictive performance over flat CAT-HCOT. This underlines the added value of hierarchical classification in a setting where classes are hierarchically structured, even if the tree-based hierarchy is shallow (7 nodes over 3 levels).

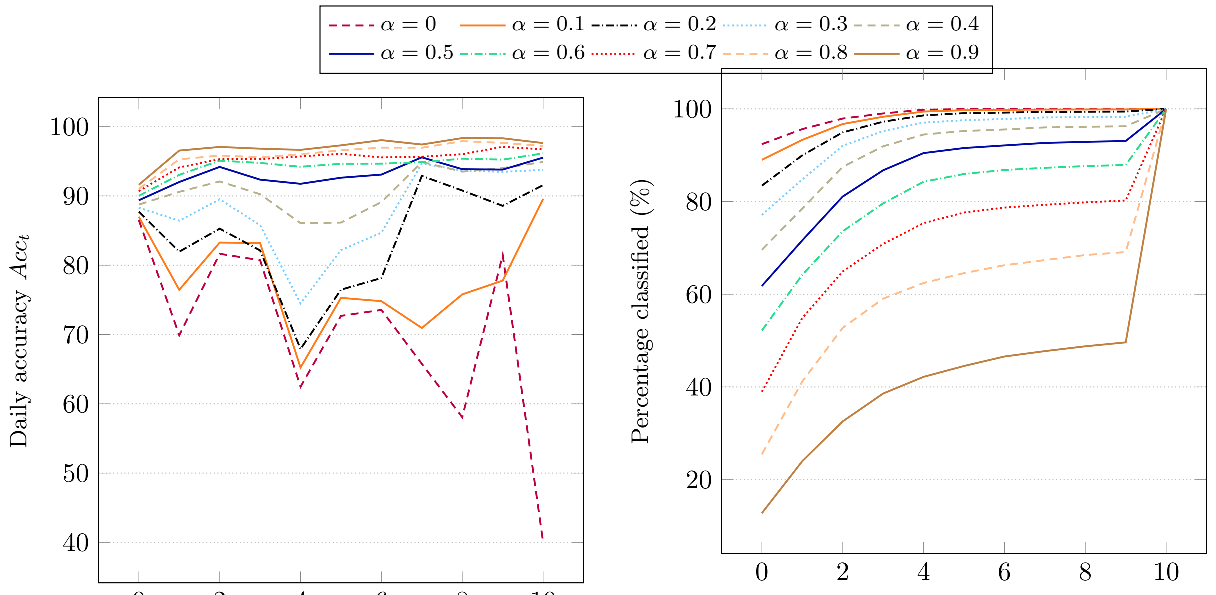
5.5. Sensitivity Analysis of Certainty Parameter

To analyse the effect of the certainty parameter α, which balances the trade-off between accuracy and timeliness, we perform a sensitivity analysis. We evaluate the CAT-HCOT algorithm for values of α ranging



(a) Daily accuracy Acc_t.
(b) Percentage of classified instances.

Fig. 11. Comparison of CAT-HCOT against static and flat baseline methods.



(a) Daily accuracy Acc_t of leaf-node predictions. **(b)** Cumulative percentage of leaf-node classified instances.

Fig. 12. Sensitivity analysis for certainty parameter α in CAT.

from 0 to 1 with step size 0.1. Figure 12 shows the $\alpha-$specific accuracy and the cumulative percentage of leaf-node predictions over time. We mention that for $\alpha = 1$, the CAT-HCOT algorithm fails, because it only gives 0.04% of instances a leaf-node prediction in the first nine days after (and including) the order date. Therefore, the results for $\alpha = 1$ are omitted from the figure. First, Figure 12a shows that higher values of α lead to higher leaf-node prediction accuracy. This make sense, because a higher certainty implies that we put more weight on accuracy than on timeliness. Second, Figure 12b shows that a higher α leads to a lower cumulative percentage of leaf-classified instances in the first nine days after the order date. This makes sense too, because, following the accuracy-timeliness trade-off, higher accuracy comes at the expense of timeliness.

Inspecting specific values of α, we find that for $\alpha < 0.5$, accuracy scores are highly volatile on the first few days after the order date. On the other hand, high fractions of instances get assigned to a leaf node on those days. The opposite story goes when $\alpha > 0.5$, for which fewer instances get leaf-classified in the first few days, but the accuracy scores of these instances reach much higher levels. Hence, in applications where timeliness is more important than accuracy, lower values of α are more suitable, while higher values of α are preferable in case accuracy is considered to be more important than timeliness.

6. Conclusions and Future Work

In this paper, we have designed a method to accurately predict the quality of product orders at a large e-commerce platform in the Netherlands shortly after the orders have been placed. In this prediction task, we face a clear trade-off between accuracy and timeliness: on the one hand, product orders should be classified as accurately as possible to ensure reliable predictions, and on the other hand, product orders should be classified as soon as possible to ensure actionable predictions. We use our own hierarchical classification method that classifies instances over time in a top-down, non-mandatory leaf-node prediction procedure with certainty-based automated thresholds.

Our results suggest that HCOT combined with the CAT algorithm, in short CAT-HCOT, achieves highly promising predictive performance. CAT-HCOT obtains a global predictive accuracy of 93.7%, with daily accuracies consistently exceeding 90%. In addition, both global and class-specific hierarchical performance measures almost always achieve scores above 90%. With respect to timeliness, CAT-HCOT is able to classify around 40% of orders on the same date these orders are placed, approximately 80% of orders within 5 days after the order date, and, by design, 100% of orders after 10 days. All in all, we find that CAT-HCOT adequately captures the accuracy vs. timeliness trade-off by using a certainty-based classification approach, while a static method, which does not incorporate certainty, does not allow for such a balanced trade-off. We further find that CAT-HCOT achieves major improvements over a flat baseline method in terms of hierarchical precision and recall scores on the most specific and least occurring Deficient classes, which illustrates that taking into account the hierarchical class structure improves predictive performance. Last, CAT-HCOT is widely applicable and highly flexible, because the certainty parameter α can be user-specified to balance the trade-off between accuracy and timeliness.

The main contribution of our research lies in three areas. First, we propose a new method of using hierarchical classification in a dynamic way. Second, we introduce a new certainty-based automated thresholds algorithm and combine this algorithm with hierarchical classification over time. Last, we add to the literature on hierarchical classification by applying this technique in a shallow-tree setting with imbalanced data, specifically tailored to the field of company satisfaction in e-commerce.

In the future, this research may be extended into various interesting directions. First, we point out that, although this research is tailored to the domain of company satisfaction in e-commerce, the CAT-HCOT framework can be applied to any other dynamic hierarchical classification problem. Furthermore, the CAT algorithm does not require the presence of a time component and can thus be applied to any classification problem that involves blocking. Second, we can implement the CAT-HCOT algorithm by using different classification methods that are able to produce near-continuous class prediction probabilities, such as Naive Bayes, Neural Networks, or Support Vector Machines. Third, it can be interesting to use a different hierarchical structure in the class labels. In this research, the tree-based hierarchy is defined based on business logic, but it might be that different splits at different tree levels lead to better classification performance. Last, instead of predicting class labels, it can be interesting to predict the determinants that determine the quality of a product order, which include cancellations, late delivery, customer cases, and returns. This would give more insights into the quality of product orders from a business perspective, providing e-commerce platforms with the reasons why a particular product order is perfect or deficient.

References

[1] A. Addis, G. Armano, E. Vargiu, Assessing progressive filtering to perform hierarchical text categorization in presence of input imbalance, in: Proceedings of the 2nd International Conference on Knowledge Discovery and Information Retrieval (KDIR 2010), SciTePress, 14–23, 2010.

[2] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS 2011), Curran Associates, Inc., 2011.

[3] J. Bergstra, D. Yamins, D. D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: Proceedings of the 30th International Conference on Machine Learning (ICML 2013), JMLR.org, 115–123, 2013.

[4] M. Ceci, D. Malerba, Hierarchical classification of HTML documents with WebClassII, in: Proceedings of the 25th European Conference on Information Retrieval (ECIR 2003), Lecture Notes in Computer Science, vol. 2633, Springer, 57–72, 2003.

[5] Y. Chen, S. Zhao, Z. Xie, D. Lu, E. Chen, Mapping multiple tree species classes using a hierarchical procedure with optimized node variables and thresholds based on high spatial resolution satellite data, GIScience & Remote Sensing 57 (4) (2020) 526–542.

[6] A. Clare, R. D. King, Knowledge discovery in multi-label phenotype data, in: Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2001), Lecture Notes in Computer Science, vol. 2168, Springer, 42–53, 2001.

[7] E. Costa, A. Lorena, A. Carvalho, A. Freitas, A review of performance evaluation measures for hierarchical classifiers, in: Proceedings of the 2nd AAAI Evaluation Methods for Machine Learning Workshop (EMML 2007), AAAI, 1–6, 2007.

[8] I. Dimitrovski, D. Kocev, S. Loskovska, S. Džeroski, Hierarchical annotation of medical images, Pattern Recognition 44 (10-11) (2011) 2436–2449.

[9] N. Donthu, A. Gustafsson, Effects of COVID-19 on business and research, Journal of Business Research 117 (2020) 284–289.

[10] S. Dumais, H. Chen, Hierarchical classification of web content, in: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2000), ACM, 256–263, 2000.

[11] C. Elkan, The foundations of cost-sensitive learning, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), Morgan Kaufmann Publishers Inc., 973–978, 2001.

[12] A. Freitas, A. Carvalho, A tutorial on hierarchical classification with applications in bioinformatics, Research and Trends in Data Mining Technologies and Applications (2007) 175–208.

[13] D. Gefen, D. W. Straub, Consumer trust in B2C e-Commerce and the importance of social presence: experiments in e-Products and e-Services, Omega 32 (6) (2004) 407–424.

[14] S. Guo, H. Zhao, Hierarchical classification with multi-path selection based on granular computing, Artificial Intelligence Review 54 (3) (2021) 2067–2089.

[15] C. Jiang, Y. Liu, Y. Ding, K. Liang, R. Duan, Capturing helpful reviews from social media for product quality improvement: a multi-class classification approach, International Journal of Production Research 55 (12) (2017) 3528–3541.

[16] A. Khalemsky, R. Gelbard, A dynamic classification unit for online segmentation of big data via small data buffers, Decision Support Systems 128 (2020) 113157.

[17] S. Kiritchenko, S. Matwin, R. Nock, A. F. Famili, Learning and evaluation in the presence of class hierarchies: Application to text categorization, in: Proceedings of the 9th Conference of the Canadian Society for Computational Studies of Intelligence (Canadian AI 2006), Lecture Notes in Computer Science, vol. 4013, Springer, 395–406, 2006.

[18] R. Martinez-Cantin, BayesOpt: a Bayesian optimization library for nonlinear optimization, experimental design and bandits, Journal of Machine Learning Research 15 (2014) 3735–3739.

[19] C. Meiring, A. Dixit, S. Harris, N. S. MacCallum, D. A. Brealey, P. J. Watkinson, A. Jones, S. Ashworth, R. Beale, S. J. Brett, et al., Optimal intensive care outcome prediction over time using machine learning, PLOS One 13 (11) (2018) e0206862.

[20] 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, E. Duchesnay, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[21] S. Ruder, P. Ghaffari, J. G. Breslin, A hierarchical model of reviews for aspect-based sentiment analysis, in: Proceedings for the 2016 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 999–1005, 2016.

[22] C. N. Silla, A. A. Freitas, A survey of hierarchical classification across different application domains, Data Mining and Knowledge Discovery 22 (1-2) (2011) 31–72.

[23] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Information Processing & Management 45 (4) (2009) 427–437.

[24] A. Sun, E.-P. Lim, Hierarchical text classification and evaluation, in: Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM 2001), IEEE, 521–528, 2001.

[25] A. Sun, E.-P. Lim, W.-K. Ng, Performance measurement framework for hierarchical text classification, Journal of the American Society for Information Science and Technology 54 (11) (2003) 1014–1028.

[26] D. Vandic, F. Frasincar, U. Kaymak, A Framework for Product Description Classification in E-commerce, Journal of Web Engineering 17 (1&2) (2018) 1–27.

[27] D. Warshaw, Decision Tree Hierarchical Multi-Classifier, https://github.com/davidwarshaw/hmc, accessed: March 2022, 2017.

[28] Z. Xiao, E. Dellandrea, W. Dou, L. Chen, Automatic hierarchical classification of emotional speech, in: Proceedings of the 9th IEEE International Symposium on Multimedia Workshops (ISMW 2007), IEEE, 291–296, 2007.

[29] W. Zheng, H. Zhao, Cost-sensitive hierarchical classification for imbalance classes, Applied Intelligence 50 (8) (2020) 2328–2338.

[30] X. Zheng, S. Zhu, Z. Lin, Capturing the essence of word-of-mouth for social commerce: Assessing the quality of online e-commerce reviews by a semi-supervised approach, Decision Support Systems 56 (2013) 211–222.

[31] L. Zhou, L. Dai, D. Zhang, Online shopping acceptance model - A critical survey of consumer factors in online shopping, Journal of Electronic Commerce Research 8 (1) (2007) 41–62.

Appendix A Hyperparameter Optimisation

To implement TPE Bayesian hyperparameter optimisation, we use Python's *hyperopt* package. As our selection function, we apply the *expected improvement* criterion, while the error that we aim to minimise is $-1 \times hF$ (such that we maximise the hF-score). In terms of hyperparameters and their parameter space, for LR we optimise over the regularisation type, i.e., either L1- or L2-regularisation. As for RF, we tune the number of tree estimators (*N trees*) and the maximum depth (*max depth*) of those trees. As prior for both hyperparameters, we apply a uniform integer distribution. The number of estimators is chosen to range between 10 and 50 (with stepsize 5), while the maximum depth of our trees is set between 5 and 15 (with stepsize 1). For CAT-HCOT, we implement hyperparameter optimisation on all 8 possible classifier combinations. This means the number of parameters to optimise ranges between 3 (LR-LR-LR) and 6 (RF-RF-RF). In the flat baseline, we either need to optimise 1 (LR) or 2 (RF) parameters. We set the maximum number of optimisation iterations to find our best hyperparameters to 20. Note, however, that in case we only evaluate LR, we do not need 20 iterations to evaluate all possible hyperparameters. Below, we present the optimal sets of hyperparameters for CAT-HCOT and the flat baseline method.

Table A.1
Optimal hyperparameters for classifier combination LR-RF-RF in CAT-HCOT (left) and either LR or RF in flat CAT-HCOT (right).

	CAT-HCOT					**Flat CAT-HCOT**		
	LR	**RF**		**RF**		**LR**	**RF**	
	Penalty[a]	Max depth	N trees	Max depth	N trees	Penalty	Max depth	N trees
Day 0	0	9	35	14	20	0*	14	45
Day 1	0	10	45	14	45	0*	14	45
Day 2	1	12	30	14	30	0*	14	40
Day 3	1	12	30	14	30	0*	14	45
Day 4	1	12	30	14	30	0*	14	40
Day 5	1	12	30	14	30	0	14*	45*
Day 6	0	10	45	14	45	1	14*	40*
Day 7-10	0	10	45	14	45	0	14*	45*

[a] 0 indicates L1-regularisation, 1 indicates L2-regularisation.
* indicates that the respective classifier is the optimal choice on the respective day.

Table A.2
Optimal hyperparameters for LR and RF in flat CAT-HCOT.

	LR	**RF**	
	Penalty[a]	Max depth	N estimators
Day 0	0*	14	45
Day 1	0*	14	45
Day 2	0*	14	40
Day 3	0*	14	45
Day 4	0*	14	40
Day 5	0	14*	45*
Day 6	1	14*	40*
Day 7	0	14*	45*
Day 8	0	14*	45*
Day 9	0	14*	45*
Day 10	0	14*	45*

[a] 0 indicates L1-regularisation, 1 indicates L2-regularisation.
* indicates that the respective classifier is the optimal choice on the respective day.