Behavior Research Methods (2022) 54:1818–1840
https://doi.org/10.3758/s13428-021-01720-4

Tracking strategy changes using machine learning classifiers

Jarrod Moss[1] · Aaron Y. Wong[1] · Jaymes A. Durriseau[1] · Gary L. Bradshaw[1]

Accepted: 29 September 2021 / Published online: 26 October 2021

Abstract

In complex tasks, high performers often have better strategies than low performers, even with similar amounts of practice. Relatively little research has examined how people form and change strategies in tasks that permit a large set of strategies. One challenge with such research is identifying strategies based on behavior. Three algorithms were developed that track the task features people use in their strategies while performing a complex task. Two of these algorithms were based on task-general, machine-learning classifiers: a support vector machine and a decision tree algorithm. The third was a task-specific algorithm. Data from several strategies in a complex task were simulated, and the algorithms were tested to see how well they identified the underlying features of the simulated strategy. The two machine-learning classifiers performed better than the task-specific algorithm. However, the two classifiers differed on how well they identified different types of strategies. The first two studies show that the ability of these algorithms to recover the underlying strategy depends on the complexity of the strategy relative to the quantity of performance data available. If the underlying strategy changes too frequently, then the performance of the algorithms suffers. However, results from the third study show that it is possible to use these algorithms to track strategy changes that occur in a task. The fourth study examines performance on data from human participants. This approach to tracking strategy exploration may enable further development of theories about how people search for and select effective strategies.

Keywords Strategy · Machine learning · Strategy change

To improve performance in complex tasks, experts develop and use task-specific strategies (Schunn et al., 2005). A strategy is a sequence of actions performed to solve a problem or accomplish a task. Strategy use and changes in strategy use have been examined in areas such as skill acquisition, problem-solving, and decision-making. However, one impediment to investigations of strategy development is that many of the tasks used to examine strategies are too complex or too simple. On one end of the spectrum are simple tasks that permit only a few strategies and lack the potential to investigate how people explore large spaces of strategies. On the other end are tasks with sufficient complexity to support a larger space of possible strategies, in which it is difficult to identify the strategies that people are using and when they are using them. Here we provide a method to identify strategies by using machine learning classifiers to analyze behavior in a task to determine the key features that are guiding the strategy-driven choices that people make.

The general theoretical framework guiding this work on identifying strategies is that people develop strategies by selecting features from the task environment to drive their decisions about what to do next (Lovett & Schunn, 1999). A task that presents several features that can be used in combination permits a large space of strategies to be developed and evaluated. Strategy choice is then based on the success, or utility, of that strategy as applied to similar problems in the past (Anderson et al., 2004; Lovett & Anderson, 1996; Lovett & Schunn, 1999). If the success rates of strategies are low enough, then the task may be re-represented to include additional features to compose new strategies. In simple tasks, there are few features to choose from, so selecting additional features can be straightforward. However, in more complex tasks, the act of identifying additional features and forming strategies based on those features is a significant problem that has received little attention.

Searching through the space of strategies can be considered a problem-solving activity where the search for a new strategy operates in a secondary problem space separate from the original task. These kinds of dual space searches have been proposed to account for rule induction and scientific reasoning

✉ Jarrod Moss
jarrod.moss@msstate.edu

[1] Department of Psychology, Mississippi State University, PO Box 6161, Mississippi State, MS 39762, USA

(Klahr & Dunbar, 1988; Simon & Lea, 1974). In a recent category induction study, Prezenski et al. (2017) presented evidence that the search for a category rule was systematic in that participants appeared to use a heuristic to generate simpler one-feature rules before more complex multi-feature rules. However, the space of possible category rules in this research was relatively small.

In prior research, the most common approach to examining strategies is to identify a critical task performance measure that can discriminate between two previously identified strategies that, in some cases, have been explicitly taught to participants. For example, in an isomorph of Luchins' water jug problems called the Building Sticks Task, the first move characterizes which of two strategies the participant is using (Lovett & Anderson, 1996; Schunn et al., 2001). In a study of Space Fortress strategy adaptation, researchers first taught and trained one flight control strategy before modifying the environment and examining the impact on the flight control strategy (Moon et al., 2013). In this case, the proportion of the time spent in a particular region of the screen determined if participants continued to use the original strategy or adopted a modified strategy.

Another approach to examine strategies is to generate data from cognitive models performing a task using various strategies and examining how human data match these models (Chen et al., 2015; Zhang & Hornof, 2014). However, developing cognitive models can require a great deal of time and task-specific knowledge. These approaches are therefore costly and are not likely to generalize well to other tasks without building a new model for each possible task strategy. Further, if a participant uses a novel strategy not implemented within the model, then this procedure cannot identify the participant's strategy.

A related line of work on the strategies that people use in decision-making tasks has led to multiple algorithms for tracking strategy use in these tasks. Many of these decision-making strategies focus on simplifying the decision, using strategies referred to as heuristics (Gigerenzer & Gaissmaier, 2011). For example, a take-the-best heuristic would only examine the most valid, or predictive, feature and ignore the rest. Several techniques have been developed that can analyze a set of decisions among a pair of alternatives and determine the heuristic being used (Hilbig & Moshagen, 2014; Lee, 2016; Lee & Newell, 2011). Most recently, a Bayesian approach has been put forward that uses multiple sources of information to identify the decision heuristic being used and can further identify heuristic changes (Lee et al., 2019).

However, there are two characteristics that differentiate these decision-making heuristic approaches from the one we describe in this paper. First, the stimuli in these decision-making studies are carefully designed to discriminate between decision-making heuristics based on the choice made on a pair of stimuli in a two-alternative choice task (e.g., Walsh & Gluck, 2016). The machine-learning approach described in this paper is tested on data that occurs naturally in a complex task in which participants select from several stimuli on any trial, and there is no guarantee it is possible to discriminate between strategies based on the choice a participant makes on any given trial because multiple strategies could yield the same choice. Second, the decision-making research focuses on identifying which of a small set of well-known heuristics such as take-the-best, weighted-additive, or tally is being used by a participant. These heuristics are focused on how participants make use of available features in a decision but not on which features are used. For example, these methods would identify the participant as using the take-the-best heuristic but are not concerned with which feature is the best one being used because the stimuli have been designed in such a way that the researcher knows which cue is the most predictive cue.

Here we describe a method that determines which combination of the many task features is incorporated in a participant's strategy while simultaneously identifying whether higher or lower values of a feature are preferred in the strategy. For example, instead of reporting that the participant is using a weighted-additive heuristic, our method reports that the participant used feature1 as the primary feature and feature2 as a secondary feature. Furthermore, increasing values on feature1 may make an option more likely to be selected, and increasing values on feature2 make an option less likely to be selected, which is referred to here as the valence of the feature. Depending on the task, determining the actual features used and how they affect decisions are both important for understanding how people search for an effective strategy. For these reasons, previous strategy identification methods are not applicable to the problem we address here.

The methods described here produce a list of feature valences ordered by importance. This list of features and their valences does not specify the exact process by which someone combines multiple features to make a decision. In that sense, it is not a perfect description of the strategy being used. However, measuring strategies in this manner defines an abstract space of feature/valence combinations in which many aspects of the similarity of strategies are captured.

Complex tasks that allow for a large space of strategies require a method for tracking the features that people are using in their strategies. This paper presents a method using machine-learning classifiers, along with a more task-specific algorithm, for tracking a participant's strategy over time as they perform a task. The process begins by training machine-learning classifiers to predict the action that a participant will take, then the trained classifier can be analyzed to identify the features used in making that prediction. The predictive features are then assumed to be the same ones that the person was using.

The goal of the present set of studies was to simulate strategies that people might use in a task with a complex space of strategies and then to evaluate the algorithms' ability to recover those strategies. The first three studies use simulations performing a task to examine whether these methods are successful at recovering the strategy features that the simulations are known to be using. The fourth study examines how these methods perform on human data. Note that only the first three studies permit accuracy values to be computed. We cannot be certain what strategies people are using as they perform the task, but we can examine how well the identified strategy explains the observed behavior in the task.

We used a modified version of the Abstract Decision Making (ADM) task (Joslyn & Hunt, 1998). The original ADM research demonstrated that it predicted performance on air traffic control and emergency dispatch tasks (Joslyn & Hunt, 1998). A modified version of the task has also been used to examine individual differences in a multitasking situation with interruptions (Bai et al., 2014). The variant used in the current research has been modified to increase the space of possible strategies that participants can use to select the next subtask to work on, and this variant is referred to as the strategic ADM (sADM).

In all variants of the ADM task, choices made in the past influence the current set of alternatives to choose from, but random factors influence the evolving task state as well. To ensure that our classifiers were accurately tracking task strategies, an ACT-R model (Anderson et al., 2004) was developed to perform the sADM task. Within the ACT-R model, a range of strategies was implemented. The ACT-R model consistently uses the strategy and therefore provides data with a known strategy for comparison with the output of strategy tracking algorithms. The complexity of the model's strategy can be controlled, and it is also possible to insert a controlled amount of noise into the model's behavior to examine the performance of the strategy-tracking algorithms in the presence of noise. This approach provides a means to evaluate the algorithms and their ability to identify the underlying strategy that generated the data.

sADM task description

The sADM task comprises two interleaved activities: selecting an object to work on and processing that object by sorting it into a bin based on its attributes. This structure mimics real-world tasks such as emergency dispatch, where there are multiple tasks one could work on and each task requires a set of actions to complete it before moving on to the next task (Joslyn & Hunt, 1998). In the sADM, an object is selected from a queue and processed by querying its attributes one at a time and then sorting it into one of four bins based on those attributes. Additional objects appear in the queue either after an object has been sorted or during the sorting process (i.e., interrupting the flow of the sorting process). Figure 1 summarizes the basic structure of the task.

The sADM task requires participants to sort objects into one of four different bins, depending on the object's attributes. Prior to beginning work on the task, participants memorize the attributes associated with each of the four bins so they can correctly sort objects. For example, bin 1 might accept only large, yellow triangles, and bin 2 might accept only small, orange octagons. The interface is text-based and controlled via five keys on the keyboard. Each object is identified by an arbitrary CVC name, and the participant must execute a series of keystrokes to query the attributes of the object before sorting. For example, a participant might select the object DAX from the example queue of objects shown in Fig. 2. After selecting DAX, the participant then presses keys to query the object's color (yellow), then another query identifies its size (large), and finally a third query to identify its shape (triangle). Based on these object attributes, which must be held in working memory, the participant now knows that the object belongs in bin 1 and can execute a series of key presses to sort it. Some objects require querying and sorting on one set of attributes (visual-based attributes) and others require querying and sorting on two sets of attributes (sound-based attributes in addition to visual-based attributes). Objects that require two levels of querying and sorting therefore require about twice as long to process.

Each task block lasts 6 min, and the goal is to score as many points as possible. In addition to the attributes that must be queried to enable sorting, objects also have a set of performance features that affect the current score. These features

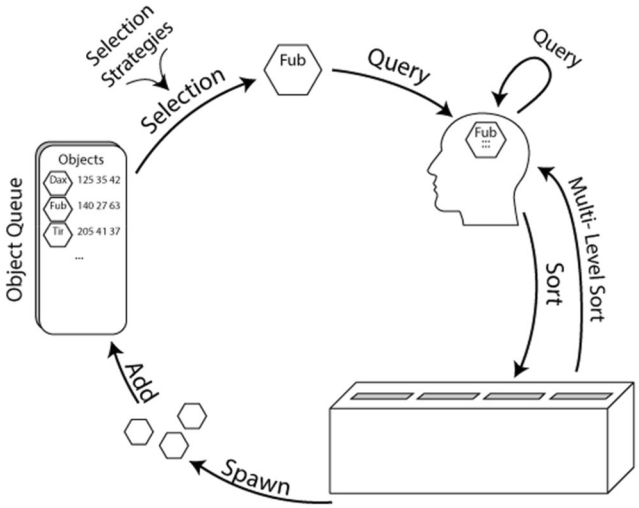

Fig. 1 A conceptual depiction of the sequence of actions in the sADM task. An object is selected for processing from the object queue. Each object has features shown in the queue that affect the performance score. Object attributes must be queried, and the results held in working memory. The object must be sorted into the bin that matches its attributes. Finally, new objects may appear during or after sorting, and the process repeats until the time limit is reached

Object Name	Time/ Deadline	Points	Penalty	Queue Position	Selection Distance
DAX	001/060	203	-45	1	2
FUB	047/120	402	-100	2	1
TIR	029/030	103	-38	3	0
WOH	015/020	153	-10	4	1
JIQ	100/110	237	-57	5	2

Fig. 2 Sample queue showing object features available in the sADM task. The last two columns do not appear explicitly in the queue that participants see; they are implicit in their position in the queue

include point values, penalties, and deadlines and are shown directly in the task interface in the queue as shown in Fig. 2. Sorting the object correctly awards the participant the point value of the object. Sorting the object into the wrong bin subtracts the object's point value from the current score. Every object has a deadline and a timer that counts down from that deadline to zero. Every time the counter reaches zero, the counter resets to the deadline and the penalty value for that object is deducted from the current score. This penalty for elapsed deadlines applies to all objects in the queue, including the object that is being processed by the participant. Therefore, performance depends on utilizing strategies that take into account multiple, dynamically changing factors (e.g., points, deadline, time until the object's next deadline).

The queue of objects initially starts with 1–3 objects, and each time an object is successfully sorted, 0–3 new objects appear probabilistically. The task adjusts the probability of new objects arriving so that the queue grows to contain approximately 7–12 objects. Objects can also occasionally arrive, as an interruption, while the participant is processing an object. When this interruption occurs, the participant must choose whether to continue processing the current object or switch to the interrupting object.

The best way to maximize performance is to sort as many high-point objects before their deadline and to prevent high-penalty objects from remaining in the queue and accumulating penalties. By manipulating the distribution of points, penalty values, and deadlines in the object queue, the task can be manipulated such that distinct strategies are needed to maximize a participant's total score. Selection strategies used to select objects from the queue are therefore critical to performance. In the queue, the participant can see the object's name, deadline, point value, penalty value, and how many seconds remain before the deadline elapses again. These object features, along with the position of the object in the queue, are the features that can guide one's selection strategy. Determining a participant's selection strategy for the sADM task is the primary focus of the strategy identification algorithms discussed here.

Strategy tracking algorithms

We developed and compared three different algorithms for extracting strategies from the sADM object-selection data. Two algorithms use standard machine-learning classifiers implemented using the scikit-learn library (Pedregosa et al., 2011): a linear support vector machine (SVM) and a decision tree (DT) classifier. A third algorithm, the Every Strategy (ES) algorithm, was implemented to compare these task-general classifiers to one that has more knowledge of the sADM queue structure. A general description of the approach taken with each of these algorithms is presented here, but the full details can be found in the Python code available at https://osf.io/qfxpr/. For a more thorough but approachable introduction to machine learning classifiers and the scikit-learn library that would be recommended to adapt the provided code to different tasks, see Géron (2019).

These selection strategy classification algorithms take as input a list of the objects on each queue presented to a participant, including all the object features, and whether each object was selected from the queue. Each sADM task block yields a list of objects that appeared in the queue for each selection that a participant made and nine features for each object: points, deadline, time until deadline, penalty value, the number of sorting levels required, whether work on the object was interrupted, position in the queue, selection distance, and the queue number. When a participant begins object selection, the middle object in the queue is initially highlighted and they move the selector up or down using key presses before selecting an object. The selection distance is the minimum number of times the selector has to be moved to reach an object.

The queue number indicates which queue the object was on when a selection was made (i.e., first, second, etc.), and it is a feature that is only used to group together all objects that appeared on the same queue. For example, there may have been nine objects present on the queue when a participant made their fourth selection. These nine objects would all have a queue number of four. The category label that the classification algorithms are being trained to predict is a binary value (selected/unselected), and the algorithms are trained to predict whether a given object would be selected or not based on the object's features. Note that these classifiers do not explicitly identify the strategy being used for object selection. A second step is needed to extract the strategy embodied in each trained classifier.

After the classifier was trained to predict which objects a participant selected, it was analyzed to determine the features present in that participant's strategy. A strategy is represented as a series of features ordered by importance and their valence. Valence here means whether higher or lower values on that feature were more likely to be selected (e.g., higher point values were more likely to be selected). For example, a strategy represented as [Points+, Deadline-] means that a higher point value was the most important feature, but the participant also preferred lower deadline values. Valence is considered because across participants or even task blocks within a participant, we have found evidence that the same feature was used with opposite valence.

Machine learning classifiers

A discussion of the details of SVM and DT classifiers is beyond the scope of the current paper, but a general characterization of the how the algorithms make a classification decision is presented because of the implications for the types of selection strategies that might best be tracked by each algorithm. Both classifiers use different techniques to learn to categorize instances of selected and unselected objects from the sADM selection data.

Because objects persist from one selection to the next (with the exception of objects selected and correctly sorted), classifiers are given distinct object states that represent the characteristics of each object as it appeared on a specific queue when a selection was made. Even though the same object name may be present in the queues for multiple selections that the participant makes, each of these is a distinct object state because the values of the object's features relative to the feature values of other objects in a queue may change from selection to selection. Each object state includes information about all features for the object (e.g., points, deadline, time in queue) that define a point in a multidimensional feature space.

Note that the object's absolute feature values are not very useful in determining whether the object will be selected or not. For example, an object with a point value of 300 might be the lowest point value in one queue or the highest in another. To simplify the learning process, features are contextualized within each queue by scaling all features within a given queue onto a 0 to 1 scale. The object in a queue with the highest point value will have a 1 after scaling, while the lowest will have a value of 0. This scaling is illustrated by the set of queues presented on the left side of Fig. 3. This method of scaling is important because these classifiers treat each object as a separate piece of data to be trained on and classified. They do not make a selection of one object from a queue of objects. Instead, each object from each queue is a separate instance to be labeled as either selected or unselected.

The SVM classifier divides up this multidimensional feature space by separating the selected and unselected objects with a hyperplane. The DT classifier builds a hierarchical set of rules to classify objects as selected or unselected (e.g., if the object has the highest point value, then it is selected, if not then if it has the lowest deadline then it is selected, otherwise it is not selected). Therefore, the DT classifier represents a participant's strategy as a sequence of binary decisions, while the SVM classifier represents the strategy as a hyperplane. The components of the DT or the location of the SVM hyperplane in the feature space can be analyzed to determine the important features of a participant's strategy. Figure 3 presents an example of this process applied to three queues in which the difference between the results of applying the SVM and DT classifiers can be seen.

Given that each queue has several objects and only one will be selected, there are more unselected than selected objects in the training data. A trivial solution is to classify all objects as 'unselected.' This solution accurately classifies all unselected objects and only makes errors on the smaller number of selected objects. To avoid this trivial solution, both the DT and SVM algorithms include mechanisms to balance the contribution of both selected and unselected objects on the resulting trained classifier using the *class_weight* parameter in the scikit-learn library for these classifiers. Conceptually, setting this *class_weight* parameter to 'balanced', as was done here, results in the classifier being penalized for a mistake using a weight that is proportional to the number of selected and unselected objects. Because there are more unselected than selected objects, the classifier is penalized more for classifying an object as unselected if it was selected by the participant than for classifying an object as selected when it was not actually selected. In other words, this weighting avoids the trivial solution of classifying all objects as unselected because the classifier is heavily penalized for classifying selected objects as unselected.

Following this feature scaling, the data are split into three cross-validation folds such that the classifier is trained on two-thirds of the data and produces a prediction accuracy on the other third. The basic unit of data is the queue number (i.e., a queue of objects). All objects from each queue are semi-

Object	Points	Deadline	Selected	Points Scaled	Deadline Scaled
Queue 1					
piyi	270	73	0	0.00	1.00
cuqi	373	52	0	0.46	0.56
zije	439	25	1	0.75	0.00
ciyi	495	60	0	1.00	0.73
Queue 2					
piyi	270	73	0	0.14	1.00
cuqi	373	52	0	0.53	0.58
ciyi	495	60	1	1.00	0.74
dago	233	30	0	0.00	0.14
lixa	253	23	0	0.08	0.00
Queue 3					
piyi	270	73	0	0.26	1.00
cuqi	373	52	1	1.00	0.58
dago	233	30	0	0.00	0.14
lixa	253	23	0	0.14	0.00
tuqu	353	64	0	0.86	0.82

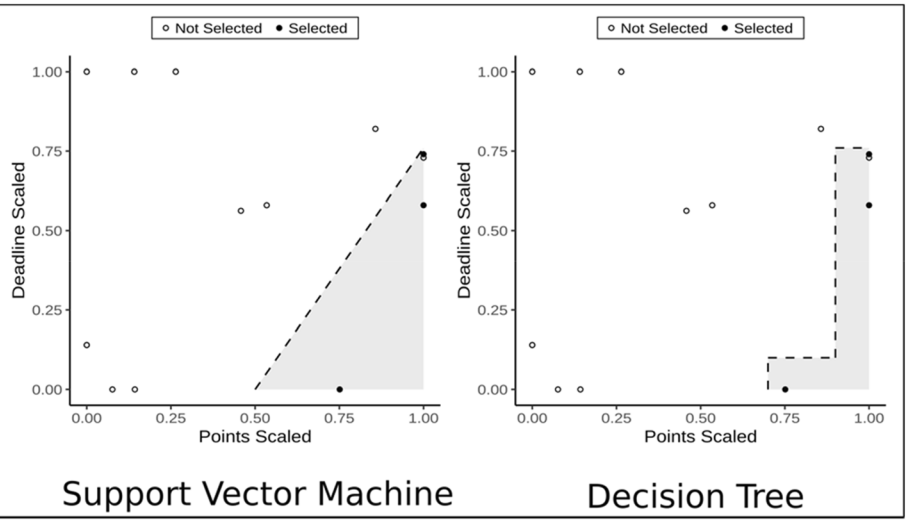

Fig. 3 A visual example of applying SVM and DT algorithms to three sample queues. On the *left* are three queues of objects from three consecutive selections such that the object selected on a prior queue is not present on a later queue with one or more new objects also appearing on the subsequent queue. The queues only include the points and deadline features for the sake of illustration. The final two columns present these two features after scaling all the features in a queue to a 0–1 scale. The two graphs plot these scaled features and illustrate how the SVM and DT algorithms divide the data into selected and unselected regions

randomly placed into one cross-validation fold so that objects from multiple selection queues are not spread across folds. The process is semi-random because the selection queues from a block are divided up into those that occur in the first, second, and third 2-min portion of the 6-min block. Each of the three cross-validation folds will contain one third of the data from each third of the block. This constraint was included so that any strategy differences that occurred over time in the block would be represented in each cross-validation fold. This process is illustrated with the training and testing folds shown in Fig. 4.

Traditional machine learning applications of these classifiers have the primary goal of maximizing prediction accuracy on novel data. However, our goal is to identify the decision-making strategy that a participant was using. Therefore, the cross-validation process was not used to maximize accuracy, but it was instead used to tune hyperparameters of the classifiers. These hyperparameters control how complex the classifier is allowed to be, which in this data translates into the number of object features used to classify the data. The more features used to classify the data means that the resulting strategy includes more object features.

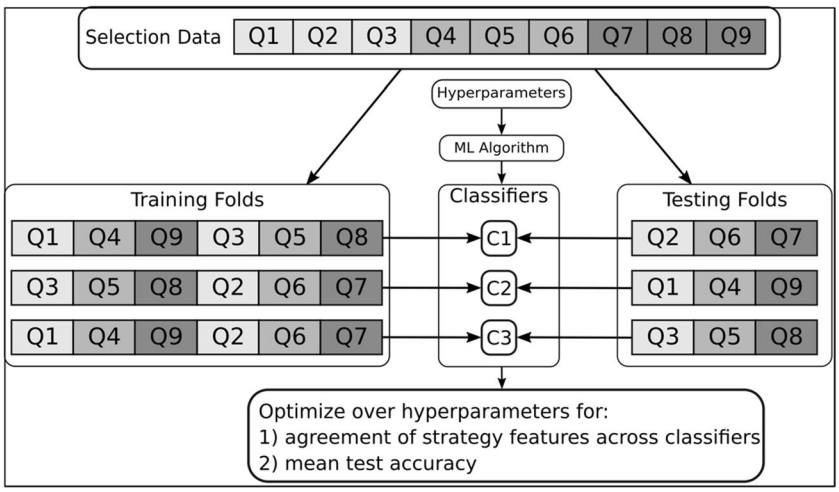

Fig. 4 In this example, a task block containing nine queues worth of object selection data is split into three cross-validation folds with consecutive queues being labeled Q1, Q2, and so on. Note that each queue contains multiple objects. *Shading* indicates from which third of the block the data come from. Three separate classifiers are trained on a training fold and then tested on the testing fold, which contains data that was not used for training the classifier. A range of values for key hyperparameters was looped over and the best-performing classifier based on a combination of accuracy on the test fold and agreement of the three classifier instances was selected as the best representation of the strategy for that task block

A grid search is performed for each hyperparameter, and the weighted average of prediction accuracy and strategy agreement is used to select the best-performing hyperparameters. Strategy agreement is determined by comparing the features extracted from each cross-validation fold, with perfect agreement occurring when all folds yield the same strategy. Again, strategies are represented as an ordered list of features and their valence. The primary purpose of this approach is to allow the complexity of the strategy to vary so long as all three folds yielded the same strategy. This approach allows for the most complex strategy supported by the data to be extracted from the data.

A good representation of a participant's strategy should lead to a prediction about which object that participant will select from a queue of objects. Because each classifier simply classifies each object in each queue as selected or not selected, the classifier might report that none of the objects were selected or that multiple objects were selected. To test the classifiers' ability to predict selection from a queue, testing accuracy was calculated by predicting which object would be selected from a queue as opposed to allowing the classifier to individually classify each object as selected or not selected. Both the DT and SVM classes in the scikit-learn library have a method that allows for a probability to be generated instead of a binary classification. For all the objects in a queue, the classifiers rank ordered the objects that would be selected according to their predicted probability of being selected. A rank order accuracy score was calculated by the formula: (queue_size - rank) / (queue_size - 1). Here queue_size is the number of objects in the queue and rank is the rank order assigned to the object that the participant picked. This rank accuracy score has a maximum value of 1 when the participant's selection matches the classifier's top ranked object and has a minimum of 0 when the participant picks the object ranked last by the classifier. The expected value of this rank accuracy score if the classifier assigned ranks randomly would be 0.5.

This rank accuracy score was used instead of a binary accuracy score so that the accuracy score would keep some sensitivity to a participant's underlying strategy even if the participant did not always pick the optimal object under a strategy. For example, a participant might pick the second highest point value because of an error or it was close enough to the maximum point value object (i.e., satisficing behavior).

Finally, the prediction accuracy, using the rank accuracy score, was compared to the accuracy expected if the algorithm had selected randomly, and a strategy is only reported if the predictive accuracy is significantly above chance levels at α = .05. This mechanism was provided to limit reporting a strategy when there is little evidence for a strategy.

DT details

The DT classifier was trained on each cross-validation fold with all features available for incorporation into the tree. An additional level of the tree can only be added if it would improve the purity of the leaf nodes by a value of N (the value of the *min_impurity_decrease* parameter in the scikit-learn implementation). This parameter essentially controls how much of an improvement in classification accuracy has to occur for adding an additional decision node to the tree. This parameter was the only hyperparameter of the DT classifier, and its value was determined as described above with a grid search ranging from values of 0.005 to 0.25. Values closer to 0 yield more complex trees. This parameter has a maximum value of 0.5, but a value of 0.25 was used for the upper bound because values above 0.25 were never optimal on any of the data (simulated or human) this approach was applied to.

The resulting DT was analyzed by examining both the importance of the object features it used for classification and the valence of those features. The importance of the features was determined using the existing *feature_importances_* attribute of the DT object. The valence is determined by analyzing the node in the tree where the feature appears to determine whether higher or lower values are associated with more selected than unselected objects. If a feature appears in more than one node on the tree, then the valence is marked as ambiguous because determining the overall impact of the feature is much more complex when it appears more than once.

SVM details

Because the SVM classifier does not have the same type of complexity-based hyperparameter as the DT, a recursive feature elimination (RFE) approach was taken. In RFE, all features are first used to train the SVM on the training set of a cross-validation fold, then the least important feature is dropped, and the process continues until there is only one feature remaining. At each iteration of the RFE process, the initial training data is again split using a three-fold cross-validation process so that a testing accuracy measure can be returned for each iteration of RFE. The classifier that is returned from this process is the one with the highest testing accuracy. For example, if the RFE process found that two features (e.g., points and deadline) led to the highest test set accuracy, then the SVM using these two features is the resulting classifier.

One problem with this RFE process is that accuracy will increase for each feature that was part of the participant's strategy, but as additional features are added, accuracy will plateau (not decrease). To obtain a pattern in accuracies with a clear peak, a complexity penalty was added to the testing method by subtracting a constant from the accuracy score for each feature added to the model. This complexity penalty parameter was used in a grid search with a range of 0.01 to 0.10. SVMs also have a C parameter that balances the margin between the data and the hyperplane and the misclassification rate. This parameter was also included in the grid search with a range of .01 to 100. As described earlier, the goal of this grid search was to

identify the classifier that produced the highest weighted average of predicted accuracy on the test set and agreement of strategy features across the cross-validation folds.

The resulting SVM was analyzed to determine the important strategy features. The absolute magnitude of the coefficients of the support vector (i.e., hyperplane) were used to rank order the importance of the features and the sign of the coefficients was used to determine the valence of the features. As a two-dimensional example, in Fig. 3, the slope of the line (i.e., support vector) provides information on the relative weighting of the points and deadline feature in the strategy captured by that SVM classifier.

ES algorithm

The DT and SVM classifiers can only be trained to classify whether a given object will be selected. The testing of these classifiers has to use a probability to pick an object from a queue as described earlier. The ES algorithm was developed as an alternative approach that learned to select one object from a queue of objects. It therefore has more task-specificity built into it.

Even though the sADM task provides the ability to explore a large strategy space, it is possible that people use fairly simple strategies. The ES algorithm uses a brute force method to determine selection strategy, but it limits the potential combinatorial problem by only considering strategies with at most two features and weighting each feature equally. For every object selection decision, the ES algorithm ranks all objects in the queue based on every possible strategy that includes one or two features. If a strategy contains two features, then the algorithm sums the rankings for the individual features to create a final ranking. The algorithm then selects the object with the highest rank. If two or more objects have the highest rank, then the algorithm selects one of the objects randomly. Similar to the other classifiers, the test accuracy for each strategy is then calculated by comparing the objects selected by the algorithm with the objects selected by the participant.

The same cross-validation approach used with the machine-learning classifiers is used with the ES algorithm. When determining which strategy was used by the participant, the ES algorithm reports which strategy had the highest rank order accuracy for the training set. This algorithm was included to see if it performed better with smaller amounts of data in some of the simulations in which noise was added.

Three parameter recovery studies were conducted to compare the algorithms. The first study examines both simple and more complex strategies using multiple features. The second study examines the performance of the algorithms when noise is added to the selection process, and the third study examines the possibility of tracking strategy changes as they happen during performance of a task.

Study 1: Evaluation of ability to detect simple and more complex strategies

This first set of simulations and analyses was conducted to assess the ability of each of the three algorithms to detect the selection strategy used by different simulations. A set of simulations was implemented using a range of strategies including single-feature strategies, a strategy that used a feature non-linearly, and two different methods of combining two features in a selection strategy. Two primary questions are addressed with these simulations. First, do all the algorithms accurately identify the strategy? Second, how much data is necessary to identify strategies in this range of complexity? This second question is relevant to establishing a lower bound on how quickly a strategy could be identified if the goal was to track which strategies participants were using while performing the task. An additional issue explored in this study is the influence of the hyperparameter values on the accuracy of the algorithms.

Method

For all the simulations, the objects that appear in the sADM task were set so that their feature values were drawn from a uniform distribution with points ranging from 50 to 550, deadline ranging from 20 to 75 s, penalty ranging from – 150 to – 35. In addition, half of the objects required one level of sorting and half required two levels. All other features of an object depend on these initial features and the dynamics of how the task unfolds as the simulated participant interacted with it. For example, time until deadline is a dynamic feature that depends on the deadline and how long the object has been in the queue.

Five single-feature variants, based on the same fundamental ACT-R model, were developed. Four of these variants always picked the highest-ranked object based on a single feature: highest points, lowest deadline, lowest time until deadline, and lowest penalty value. The fifth single-feature strategy picked the first object highlighted by the interface when going to select an object (i.e., the object in the middle of the queue).

Besides these single-feature strategies, three more complex strategy models were also implemented. First, a model that used the time until deadline feature non-linearly was implemented. If the time until deadline was 8 s or less, then lower values were preferred, but if the time until deadline was greater than 8 s then higher values were preferred. This strategy allows for objects that are nearest to their deadline to be sorted, but if the object has plenty of time until its deadline, then longer times would be preferred to allow time for handling interrupting objects if they occur during sorting. While the simulated participant always ignored interrupting objects for simplicity of implementation, human participants will often switch to interrupting objects before resuming sorting the

original object. This strategy was included as a potentially effective strategy that used a feature in a nonlinear manner.

Two other complex strategies were implemented that combined the points and deadline features in either a weighted or thresholded manner. These combination strategies were selected based on an analysis of heuristics reported in the multi-attribute decision-making literature (e.g., weighted additive, take the best) (Gigerenzer & Gaissmaier, 2011). A set of weighted combination strategies was used that produces a weighted sum of the point and deadline values. Accounting for the distribution of point and deadline values noted above, three different weighted strategies were implemented. The first weighted points and deadline equally and is referred to as the deadline=points weighted strategy. The second weighted points more than deadline in an approximately 60/40 weighting referred to as the points+deadline weighted strategy, and the third reversed this to a 40/60 weighted strategy referred to as the deadline+points weighted strategy. A range of weightings was explored to ensure that the classifiers were sensitive to a range and not one specific weighting. More extreme weightings that greatly prefer one feature over another (e.g., 80/20 points over deadline) are not likely to be distinguishable from a single-feature points strategy because of the limited number of selections in which a single-feature strategy would lead to a different selection than a strategy heavily weighted toward the points feature.

A set of threshold combination strategies was also implemented similarly to the weighted strategies where the deadline=points thresholded strategy has roughly equal contributions of both features, the deadline+points thresholded strategy has a greater contribution of the deadline feature, and the points+deadline thresholded strategy has a greater contribution of the points feature. The deadline=points strategy selected the highest point value if there was an object over 300 points (300 is the mean point value from the uniform distribution of 50–550), and if there were no objects above that threshold, then the lowest deadline object was selected. The deadline+points strategy selected the highest point value if there was an object over 250 points, and if there were no objects above that threshold, then the lowest deadline object was selected. The points+deadline strategy selected the highest point value if there was an object over 400 points, and if there were no objects above that threshold, then the lowest deadline object was selected. Just as in the weighted strategies, there are some thresholded strategies that could be indistinguishable from a single feature strategy. For example, a strategy that selected based on points as long as there was an object worth more than 100 points on the queue would almost always select the object with the highest point value. Given that each object has a value ranging from 50 to 550 points, the probability that all objects in the queue fall in the 50-to-100-point range is small.

These variants were examined to ensure that the strategy identification techniques could correctly recognize distinct strategies at different levels of complexity. All strategies were simulated for 60 6-min blocks of the sADM task. Each block was an independent set of data to detect the selection strategy being used by the model.

Results

The classifiers were assessed on two primary measures. First, as described earlier, a feature importance analysis can be done on each classifier to determine the features and their valence (i.e., are higher or lower values for a feature preferred). A simple binary scoring measure was used to assess the accuracy of the classifiers in determining the actual features used in the strategy. If the results of the classifier feature analysis process returned only the correct features and their correct valence, then the strategy was determined correctly. Otherwise, the strategy was not determined correctly. The proportion of blocks for which the correct strategy was identified was the primary measure used to examine the classifiers' ability to determine the correct strategy and is referred to as strategy feature accuracy.

A second measure was the mean prediction accuracy of the classifier on the test set during the three-fold cross-validation process used to train the algorithms. While the strategy feature accuracy measure described in the last paragraph focuses on whether the underlying features present in the strategy are identified, this prediction accuracy measure assesses whether the object that was selected from the queue of objects can be accurately predicted by the classifier. For all the objects in a queue in the test set, the classifiers rank ordered the objects that would be selected according to their predicted probability of being selected. This rank order score, defined earlier, was the measure of predictive accuracy, and is the only measure available when applying the classifiers to human data because their strategy is not known.

The mean prediction accuracy and proportion of time the correct strategy was identified for each strategy for each algorithm are shown in Table 1. For the single-feature strategies, all three classifiers performed at ceiling. Prediction accuracy was occasionally below 100% because there were infrequent ties in the data (e.g., two objects have the same point value). For the two-feature strategies, the DT performed the best for the strategy which involved combining the two features with a threshold (e.g., if there is an object over 400 points, then use points, otherwise use deadline). The SVM performed better than the DT for the weighted combinations of features. Finally, the DT performed best for the nonlinear points strategy where extreme high and low point values were preferred over mid-range point values. Based on these results and the underlying compatibility in the implementation of the DT and SVM classifiers, the outputs of these classifiers can be

Table 1 Mean rank accuracy and accuracy at recovering the strategy features for each algorithm

Strategy	DT		SVM		ES		Combined DT/SVM
	Rank Acc	Feature Acc	Rank Acc	Feature Acc	Rank Acc	Feature Acc	Feature Acc
Deadline	1.00 (.002)	1.00	1.00 (.002)	1.00	1.00 (.002)	1.00	1.00
First item	1.00 (0)	1.00	1.00 (0)	1.00	1.00 (0)	1.00	1.00
Penalty	0.99 (.005)	1.00	1.00 (.004)	1.00	1.00 (.004)	1.00	1.00
Points	1.00 (.003)	1.00	1.0 (.001)	1.00	1.00 (.001)	1.00	1.00
Time until deadline	0.94 (.03)	1.00	0.94 (.05)	1.00	0.94 (.04)	0.97	1.00
deadline = points threshold	0.96 (.03)	0.93	0.87 (.05)	0.07	0.87 (.04)	0.20	0.93
deadline + points threshold	0.96 (.03)	0.92	0.90 (.05)	0.03	0.88 (.06)	0.03	0.92
points + deadline threshold	0.95 (.03)	0.85	0.90 (.04)	0.12	0.88 (.05)	0.12	0.85
points = deadline weighted	0.87 (.05)	0.93	0.95 (.02)	0.97	0.90 (.03)	0.58	1.00
deadline + points weighted	0.87 (.04)	0.73	0.94 (.03)	0.75	0.90 (.03)	0.45	0.92
points + deadline weighted	0.86 (.05)	0.75	0.94 (.04)	0.82	0.90 (.04)	0.65	0.93
time until deadline nonlinear	0.85 (.06)	0.87	0.68 (.02)	0	0.53 (.08)	0	0.87

Note. Standard deviations are in parentheses. The combined classifier is shaded to highlight that it performs at least as good as any of the other classifiers

combined and the classifier with the highest prediction accuracy can be chosen to determine the strategy. The last column of the table shows strategy feature accuracy when the outputs of these two classifiers are combined. This combined algorithm can sometimes yield feature accuracy values above or below the performance of DT or SVM algorithms alone because the algorithm with the highest predictive accuracy of each block is selected and not the algorithm with the highest feature recovery accuracy because only the predictive accuracy would be available for human data. For example, the combined classifier generally does better at detecting both features of the weighted strategies than the SVM or DT alone.

The proportion of time the correct strategy was identified and predictive accuracy both focus on assessing the performance of the classifiers for the different strategies tested. A secondary aim of this set of simulations was an assessment of how much data would be needed to accurately identify the strategy a hypothetical participant was using in the task. The approach used here was to run the classifiers only on the first N object selections of each block of the task. The total number of selections in a block of data ranged from 36 to 39, and therefore N was varied from 15 to 35 in increments of five and whether the correct strategy features were identified was examined for each N.

For the single-feature strategies, all classifiers extracted the correct features 95–100% of the time with only the first 15 selections and all reached 100% with 20 selections. For the more complex strategies, Table 2 shows the proportion of time the correct strategy was identified for each amount of data for each classifier. The nonlinear strategy is not shown in Table 1 for the ES or SVM because they never identified the correct strategy. From Table 2, it appears that all classifiers benefit from increasing amounts of data on the more complex strategies with performance leveling out around 30–35 selections.

The final issue examined the sensitivity of the DT and SVM classifiers to changes in hyperparameters. In particular, the algorithms use a grid search over a space of hyperparameters to find the value of the hyperparameters that maximize prediction accuracy and agreement on the strategy features across cross-validation folds. Because the eventual goal is to examine human data with these algorithms and strategy feature accuracy is not available on human data from the task, this analysis focused on how prediction accuracy was affected by the hyperparameter values. Strategy feature accuracy is also shown to demonstrate that these two measures are often correlated. For the DT classifier, the minimum impurity hyperparameter value is plotted against accuracy for the simple strategies in Fig. 5 and the more complex strategies in Fig. 6. There appears to be an optimal range for this hyperparameter between 0.025 and 0.125. The SVM had two hyperparameters (C and penalty). The SVM strategy accuracy results are plotted for these hyperparameter values in Fig. 7 for the simple strategies and Fig. 8 for the complex strategies. These figures show some hyperparameter values do not perform well, but there is a large space of hyperparameters that perform well in both classifiers.

Discussion

For simple, monotonic, single-feature strategies, all the classifiers performed well, requiring minimal data and showing little sensitivity to hyperparameter values. However, significant differences emerged when slightly more complex strategies were examined. Two methods of combining multiple features were examined: threshold and weighted. For the

Table 2 Strategy feature accuracy by number of selections included in the data

Classifier	Strategy	Number of Selections				
		15	20	25	30	35
DT	deadline=points threshold	0.20	0.48	0.78	0.88	0.92
	deadline+points threshold	0.50	0.80	0.90	0.90	0.93
	points+deadline threshold	0.07	0.12	0.42	0.52	0.80
	deadline=points weighted	0.23	0.48	0.58	0.73	0.83
	deadline+points weighted	0.12	0.38	0.58	0.68	0.75
	points+deadline weighted	0.12	0.40	0.53	0.63	0.80
	time until deadline nonlinear	0.28	0.58	0.78	0.78	0.80
SVM	deadline=points threshold	0.02	0.05	0.12	0.13	0.20
	deadline+points threshold	0.07	0.08	0.03	0.03	0.05
	points+deadline threshold	0.02	0.02	0.02	0.07	0.08
	deadline=points weighted	0.50	0.73	0.78	0.87	0.85
	deadline+points weighted	0.40	0.53	0.65	0.68	0.67
	points+deadline weighted	0.55	0.73	0.72	0.78	0.85
ES	deadline=points threshold	0.05	0.05	0.10	0.15	0.13
	deadline+points threshold	0.03	0.07	0.02	0.02	0.07
	points+deadline threshold	0.00	0.02	0.03	0.10	0.10
	deadline=points weighted	0.37	0.43	0.45	0.58	0.62
	deadline+points weighted	0.20	0.38	0.45	0.45	0.48
	points+deadline weighted	0.35	0.45	0.57	0.62	0.67
Combined DT/SVM	deadline=points threshold	0.18	0.48	0.78	0.88	0.92
	deadline+points threshold	0.52	0.80	0.88	0.88	0.93
	points+deadline threshold	0.05	0.10	0.40	0.52	0.82
	deadline=points weighted	0.57	0.75	0.85	0.90	0.92
	deadline+points weighted	0.38	0.62	0.75	0.80	0.88
	points+deadline weighted	0.55	0.78	0.80	0.83	0.95
	time until deadline nonlinear	0.28	0.58	0.78	0.78	0.80

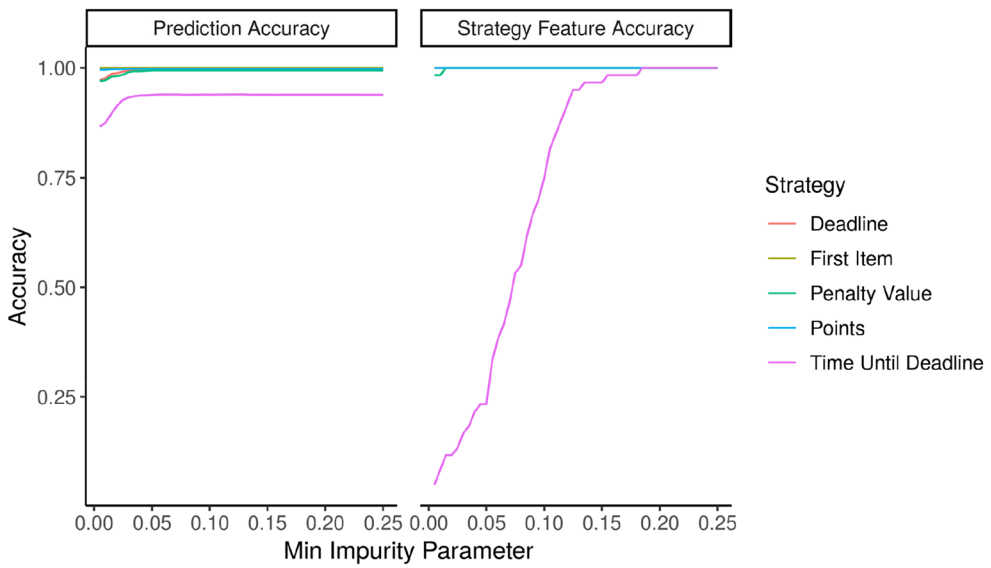

Fig. 5 Mean prediction accuracy for the DT hyperparameter for the single-feature strategies

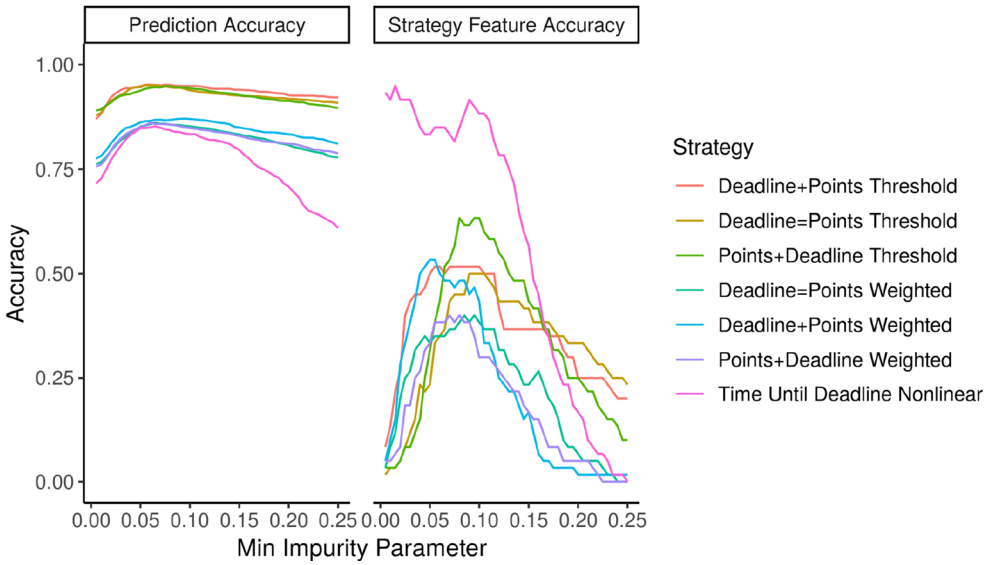

Fig. 6 Mean prediction accuracy for the DT hyperparameter for the more complex strategies

threshold strategy, the DT classifier performed the best, and for the weighted strategy, the SVM classifier performed the best. This result is consistent with how the two underlying data structures represent the classification process.

The DT creates a tree of if-then rules to correctly classify an sADM object as selected or not, while the SVM separates selected and unselected objects with a hyperplane. The hyperplane in a linear SVM in two dimensions is a line that is equivalent to a weighting between two features. This finding supports the expectation that the match between the strategy and the type of classifier is important. The more similar the classifier's representation of the strategy is to the strategy in the simulation, the better the classifier captures the strategy.

The nonlinear strategy was also designed to be something that a linear SVM or the ES strategy should not be able to capture, and the only classifier that performed well on this strategy was the DT. It is possible that a nonlinear SVM might do well on this strategy, but then the problem becomes analyzing the SVM to provide a human-readable representation of the strategy.

Given these results, it should be possible to take a multi-classifier approach to strategy identification. For example, the DT and SVM classifiers could both be used on a set of data, and then the cross-validated prediction accuracy could determine which of the two classifiers should be analyzed to identify the strategy. In the data examined in this study, the combination of

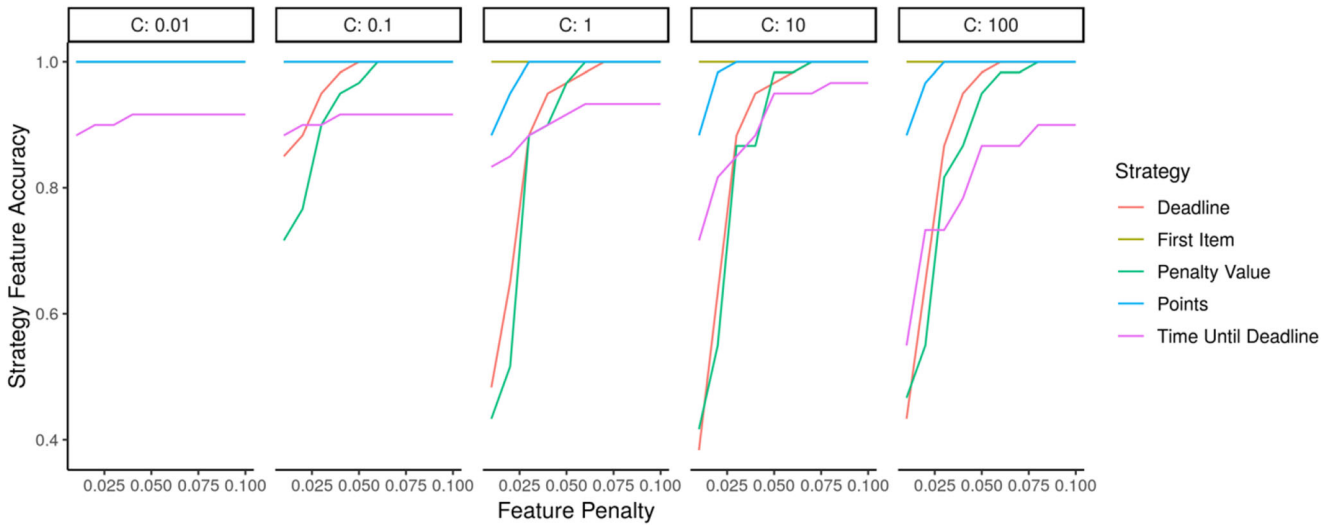

Fig. 7 Mean prediction accuracy for the SVM hyperparameters for the single-feature strategies

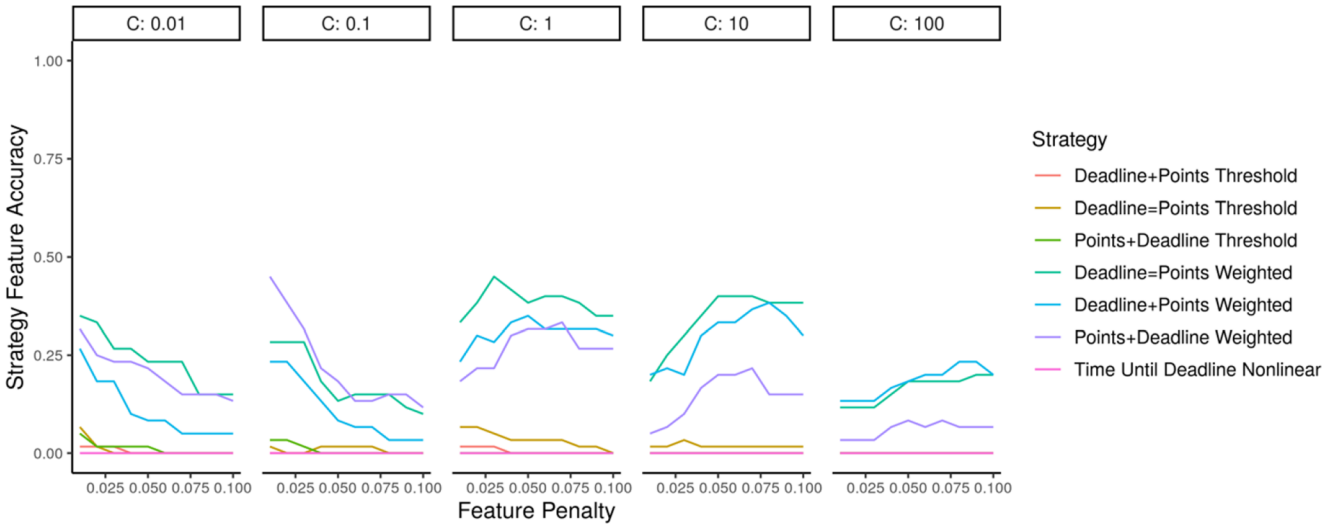

Fig. 8 Mean prediction accuracy for the SVM hyperparameters for the more complex strategies

the DT and SVM classifiers performed well on all strategies examined.

One limitation of this study is that the ACT-R model was perfectly consistent in its adherence to a given strategy. However, people are likely to deviate from this level of consistency, introducing noise into the data. The second study addresses this possibility by examining two different types of noise that can be introduced into the data to determine how sensitive the classifiers are to noise.

Study 2: Strategy detection in the presence of noise

This second set of simulations examines how well the strategy detection algorithms handle noise in the selection data. People may not always select the object that perfect execution of a selection strategy would demand. Lapses of attention or pressing the wrong key will lead to selections that deviate from the ideal selection strategy. Two different types of noise were simulated to examine the performance of the algorithms in less-than-ideal circumstances.

First, random noise was added to a single-feature strategy at varying levels by having the model select according to the strategy at times and randomly other times. Levels of randomness were varied from 10 to 90% of selections being random. In addition, random selections were added to the more complex two-feature selection strategies with levels of randomness ranging from 10 to 50% of selections. A second type of noise was also examined in which the model simulated a form of satisficing. For example, under a points strategy, instead of always selecting the highest point value object, the satisficing version of the strategy just picked an object that had a sufficiently high point value (e.g., within 50 points of the highest point value in the queue).

Method

All aspects of the simulation method were identical to Study 1 except for introducing noise into the simulated selection strategies. Random noise was added to the points selection strategy used in Study 1 at varying levels by manipulating the utility of two productions controlling the choice between the points strategy and a strategy that randomly selected an object. The result was that it was possible to control how often the model selected randomly or selected based on the points strategy. Simulations were run with random selections being made 10, 30, 50, 70, or 90% of the time. In addition, a model was run with the random selection strategy only to determine if the classifiers would correctly report that no strategy was being used.

Random noise was also added to the deadline=points weighted and deadline+points threshold strategies from Study 1. For these two strategies, selections were random 10, 30, or 50% of the time. Higher levels of noise were not examined with these more complex strategies because it was expected that even moderate amounts of noise would make the two-feature strategies difficult to identify. In addition, given the lack of large differences between different weightings or thresholds in Study 1, only these two examples of the complex strategies were used in this study.

Finally, a satisficing version of the points strategy was used in which the model selected an object within 50 points of the highest object in the queue. For example, if the queue contained objects worth 100, 200, 325, and 350 points, then this satisficing strategy would have selected either of the latter two objects with equal likelihood. A similar satisficing strategy was also implemented for the deadline=points weighted and deadline+points threshold strategies. For the thresholded combination, the satisficing object was within 50 points if points was used or 5 s if deadline was used. For the weighted combination, the satisficing object was withing 10 of the weighted combination of the features.

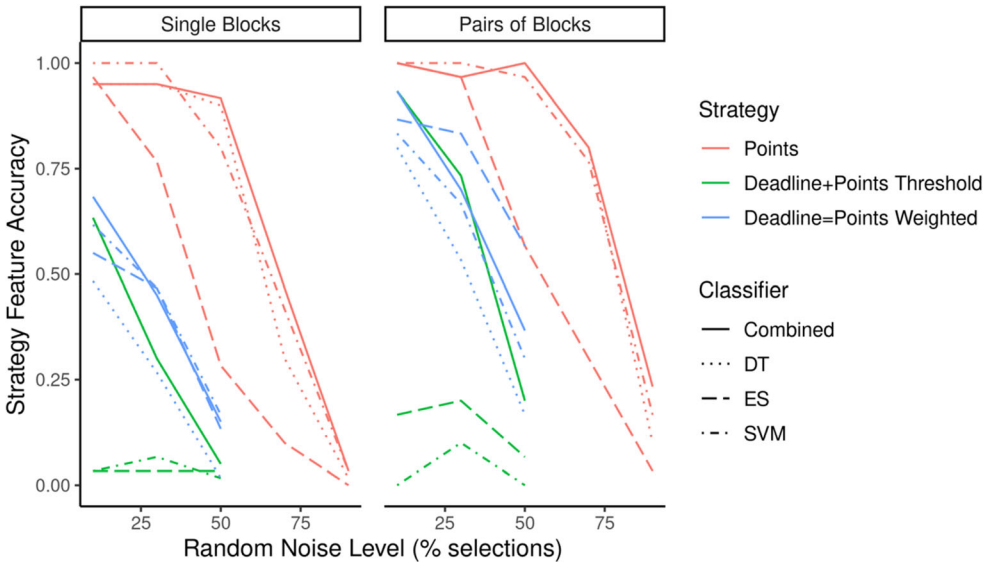

Fig. 9 Strategy feature accuracy with varying levels of random noise. For the threshold strategy, the DT and combined lines are identical

Results

The accuracies for detecting the correct features for the simple and complex strategies with noise are shown in the left half of Fig. 9. The predictive accuracies using the rank accuracy metric are shown in Fig. 10. For the simple strategy, most of the classifiers performed well until the noise level exceeded 70%. However, they did not fare as well on the more complex strategies with noise. Performance fell into the 60–70% range even with only 10% random noise. The ES and SVM performed the best on the weighted strategy, while the DT performed better on the threshold combination strategy. When the classifiers did not identify both features correctly, the majority of the time (> 75%) they identified one of the two features. The combination of the DT and SVM classifiers performed as well as either of the DT or SVM alone on all strategies.

The lower accuracies might be due to needing more data to detect the strategy in the presence of noise. To test this possibility, the data were aggregated into 30 pairs of blocks instead of 60 single blocks, and the classifiers were run on these pairs of blocks. The results in the right half of Figs. 9 and 10 show higher accuracies for pairs of blocks, which is consistent with the idea that the classifiers could still detect the more complex strategies with some random noise given additional data.

While random noise interfered with strategy detection, the classifiers did about the same with satisficing strategies as they did with the smallest amount of random noise. These results are presented in Fig. 11, and the predictive accuracies using the rank accuracy metric

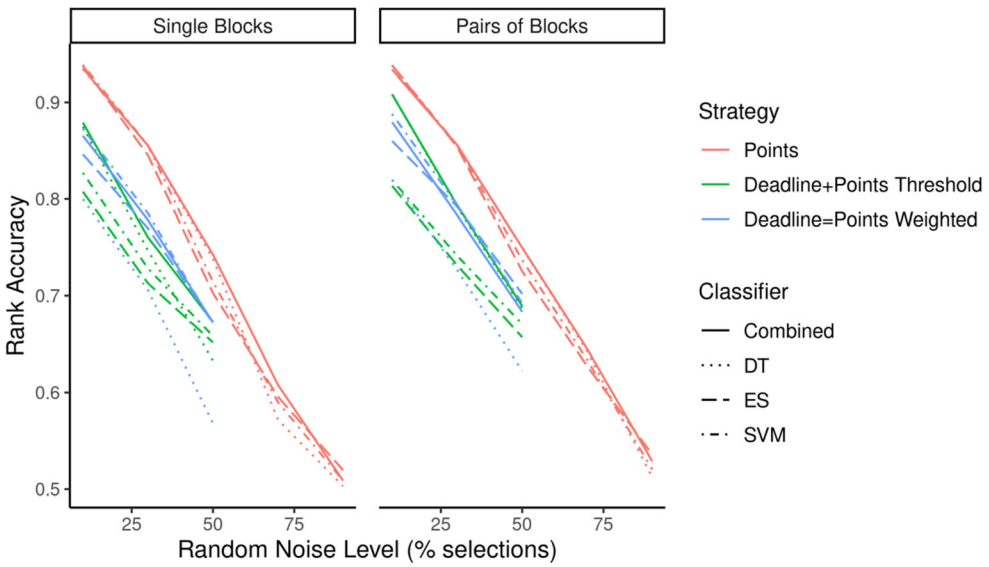

Fig. 10 Predictive rank accuracy with varying levels of random noise

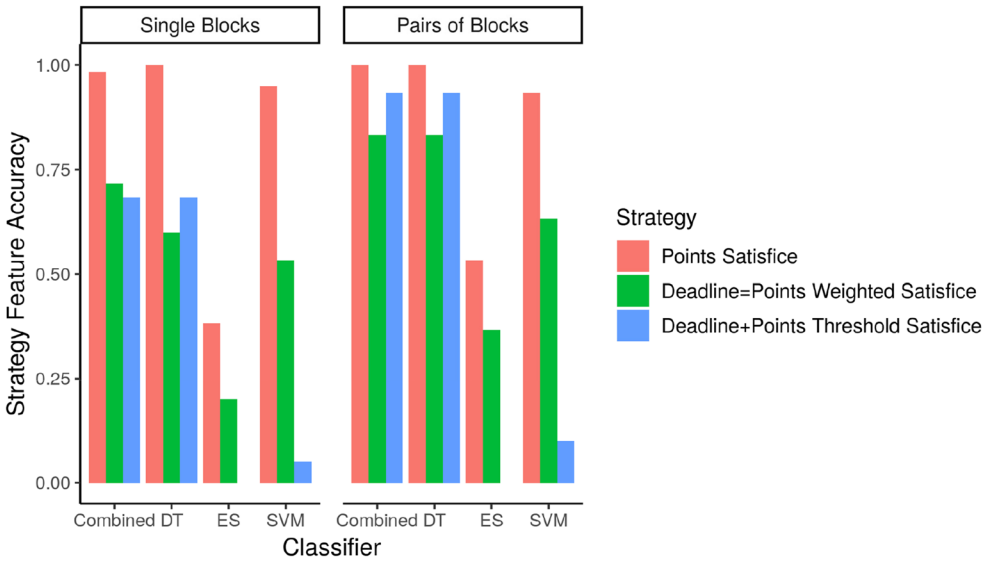

Fig. 11 Mean strategy feature accuracy with satisficing strategies

are shown in Fig. 12. Again, the right half of the figures show that with more data, the classifiers perform at higher levels, approaching performance on the complex strategies without noise.

On the random strategy, all classifiers correctly reported that there was no apparent strategy between 95 and 100% of the time. This range is to be expected given that the mechanism for detecting whether there is a strategy is whether there is evidence that the prediction accuracy is above chance rates with $p < .05$.

An analysis of the influence of the hyperparameters was also carried out, as in Study 1. The DT accuracy results were similar, showing the best accuracies for the minimum impurity increase parameter between .025 and .125. The SVM results were also similar. For the simple strategy, penalty decreased accuracy at low values of C, but it increased accuracy at high values of C. The complex strategies were only above 0 at values of C of 10 or less with higher accuracy at lower values of the penalty.

Discussion

For a single-feature strategy, most of the classifiers performed well even with up to 70% of the selections being made randomly. The more complex two-feature strategies were more difficult to identify even with relatively low levels of random noise. However, a more realistic form of "noise" in human selection data may arise from satisficing. In the satisficing models, a "good enough" object consistent with the strategy can be selected even though it may not have been the optimal

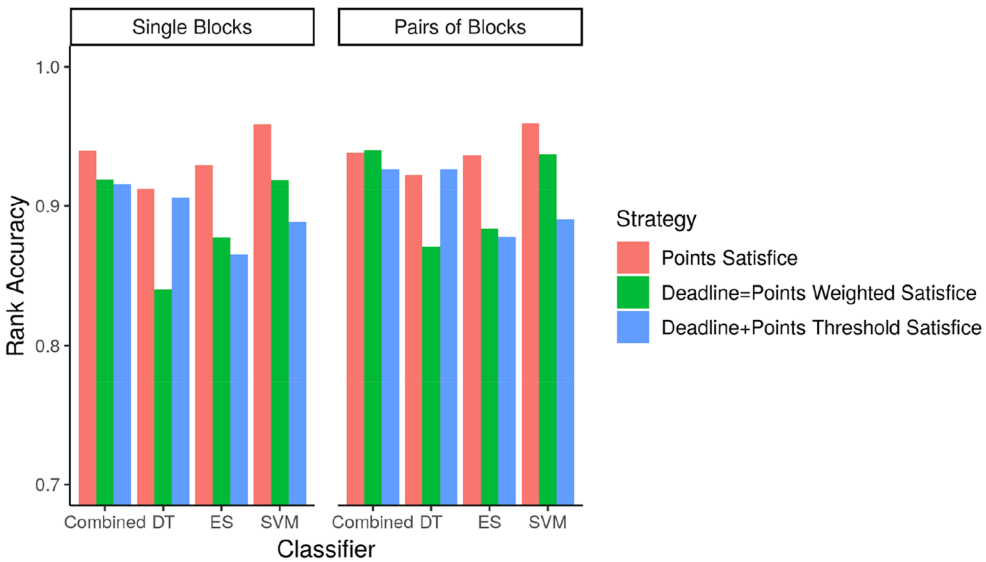

Fig. 12 Mean predictive rank accuracy with satisficing strategies

object. This kind of behavior might arise in people from taking a satisficing approach or by making an error in picking the optimal object. Under these satisficing strategies, the combined DT/SVM classifier always identified the correct single-feature strategy and did so about 70% of the time with two-feature strategies.

The classifiers all benefitted from having more selection data when identifying the two-feature strategies under the presence of noise. However, this was only true for strategies that the classifiers can represent. As noted before, the SVM and ES classifiers cannot represent the threshold version of the two-feature strategy well, so providing additional selection data did not help in these cases.

Another finding was that a similar range of hyperparameters yielded the best-performing classifiers in both this study and the first study. This result has the practical effect that a smaller range of parameters could be examined in the grid search, and this reduction would lead to reduced computation time in running the classifiers. This computational consideration would be most important when the classifiers were used to identify strategies in real time during performance of the task. For example, it would be possible to identify a participant's strategy and modify the task or intervene in some other way, depending on the purpose of the study.

Based on the results presented so far, it is possible to formulate some general recommendations for use of this method. The combined DT/SVM classifier generally performs as well as or better than any of the other approaches. As can be seen in Figs. 9 and 11, there are a few instances where a single classifier does perform better but the combined classifier is always within 5% of the maximum strategy feature accuracy in these cases. Therefore, this combined classifier as described here and documented in the available code would be the recommended approach. Another recommendation is to apply the method to tasks with at least 30–40 decision trials. If strategies using two or more features in the presence of significant selection noise or satisficing is anticipated, then increasing the number of decision trials will likely yield better results (e.g., using pairs of blocks as in results shown Figs. 9 and 11).

Cutoffs should be established based on the classification accuracy reported by the classifier so that a researcher can be reasonably confident that participant strategies have been identified accurately. Figure 13 shows a comparison between the classification accuracy of the combined DT/SVM classifier and the proportion of time all features for a tested strategy were correctly identified. This figure includes all strategies tested in Studies 1 and 2. The classifier is highly accurate without noise, and a cutoff of 0.80 accuracy results in at least 67% of the blocks having all features correctly identified. Below this cutoff, it is more likely that the classifier only identified some of the features in the strategy.

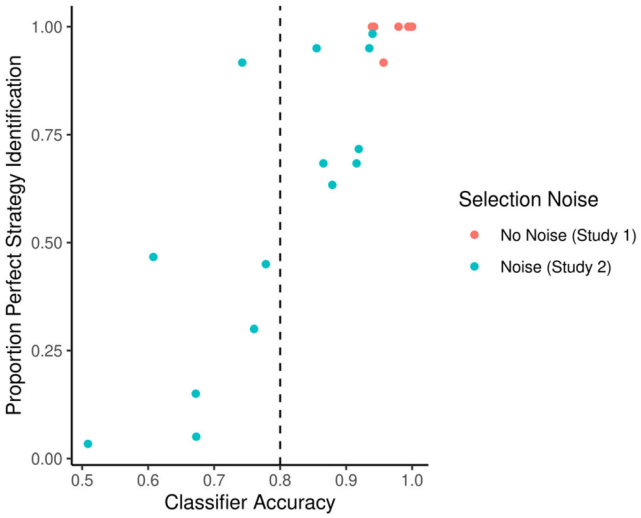

Fig. 13 Proportion of blocks where the combined DT/SVM classifier correctly identified all features in a strategy by the classifier's cross-validated accuracy. When applying the classifier to human data, only the cross-validated accuracy is known. The *dashed line* corresponds to a recommended threshold for estimating when the strategy has been correctly identified

This approach to examining strategies based on task data will be successful to the degree that the strategies people are using are simple and consistent enough to be captured by the amount of data available in the task. One form of inconsistency arises because people might change their strategy within a block of data. The next study examines the possibility of running a modified version of the SVM and DT classifiers over the time course of one block in which one or more strategy switches take place.

Study 3: Strategy detection over time

This set of simulations examined modified versions of the SVM and DT classifiers that continuously predict future selections as new selections occur over time. This allows a determination of whether the classifiers can detect a change of selection strategy within a block of the sADM task. Another purpose is to explore the possibility of running these strategy identification algorithms concurrently with the task in order to allow for interventions in the task based on the detected strategy. For example, it might be useful to prompt a participant that the strategy being used is not a high-performing strategy or that a strategy change just made will lead to worse performance.

The primary issues investigated in this study are the complexity of the strategy that can be identified and the frequency of strategy switches that can be tracked. As shown in the prior studies, detection of a single-feature strategy is relatively easy

and requires a minimal amount of data. However, a two-feature strategy requires more data, and it will therefore be more difficult to track more complex strategies if frequent strategy switches occur. In this study, we examined 1–3 strategy switches during a block of the sADM task with one- and two-feature strategies to address how strategy complexity and switching frequency impact the ability to identify a strategy.

Method

The DT and SVM classifiers were modified to identify strategies based on selections that occurred over time. For every selection within a block, the classifiers would attempt to identify the strategy from prior selections within the block and would then predict the next few selections. Instead of the three-fold cross-validation process used in the first two studies, the training fold was composed of past selections and the testing fold was composed of the next five selections. Five testing selections were used as a compromise between being able to detect strategy shifts quickly and being able to provide some limited range of prediction accuracy values to be used in identifying the best-performing set of features.

As an example of this tradeoff, consider the point in a block after 20 selections have occurred and in which a strategy change occurred on the 21st selection. With a testing window of five samples at the 20th selection, then the first 15 selections can be used to train the classifier and the next five for testing. These selections used the same strategy so the classifier would have a very good chance of detecting a single-feature strategy with 15 selections to train on. The classifier's performance is assessed on selections 16–20 and performs well, so a stable strategy is reported. However, starting on the 21st selection, a strategy change has occurred. The classifier now has 16 training selections and five testing instances (one of which will not be predicted well by the prior strategy). Fast forwarding to the 25th selection, the classifier now has 20 training selections with the old strategy and five testing selections with a new strategy. Prediction accuracy will be very low, and a strategy change could be identified because of the decrease in accuracy. With a shorter testing window (e.g., 3), the strategy change could have been identified earlier. However, the classifier might often perform reasonably well on three selections just by chance, making it more difficult to have confidence in the strategy reported by the classifier. Conversely, with a testing window of ten selections, it will take longer to identify a strategy switch and a switch occurring at the end of a block might not be detected at all. For this reason, a testing window of five selections was used as a reasonable tradeoff between these considerations.

As in the earlier studies, a grid search was used to identify the best-performing classifier parameters. The classier-specific hyperparameters (e.g., min impurity decease for DT) and the training window size (5, 10, 15, 20, 25, and 30 past selections) were used. Larger training windows increase the training data, which should increase the odds of identifying the correct strategy if the strategy did not change. However, after a strategy change, a smaller training window should perform better because it is less likely to contain selections from two strategies. The same ranges of hyperparameters were used as in the prior two studies. All of the details can be found in the code for these simulations on the OSF site at https://osf.io/qfxpr/.

In Studies 1 and 2, the hyperparameters were selected based on a weighted average of strategy agreement across cross-validation folds and prediction accuracy. However, without multiple folds, strategy agreement across multiple folds cannot be computed, so the classifiers maximized test set accuracy along with a 5% complexity penalty for every feature included after the first. This complexity penalty helped to prevent overfitting to the training data by making sure that any feature added to the strategy had to lead to an increase in prediction accuracy of at least 5%.

The strategies implemented in the simulations included one, two, or three strategy switches at equally spaced intervals, all between single feature strategies. For example, the points-deadline switch model switched from a points-driven selection strategy to one where the lowest deadline is selected at the mid-point of the block. A points-deadline-penalty strategy was examined for the two switches, and a points-deadline-points-deadline strategy was examined for the three switches. In addition, another model was implemented that started with a points+deadline weighted strategy and switched to a points strategy. This final model was used to examine whether a two-feature strategy could be identified with only a small set of training instances prior to a strategy switch. It was expected based on the results of Study 1 that this would be challenging to do with little training data. Each of the strategies was simulated for 60 blocks of sADM task performance, as in the prior studies. Given that the ES did not perform better than the SVM or DT strategies, it was not modified to identify strategy changes over time.

Results

To investigate the frequency of strategy switches that could be identified, data from three different strategy switching frequencies were simulated. These strategies were always simple single-feature strategies. The principal outcome measure was whether the feature was correctly identified at each selection within a block. The top left panel of Fig. 14 shows that all classifiers did well at identifying a single strategy switch by correctly picking up on the correct strategy feature both before and after the switch. As in the prior studies, a combination of the DT and SVM classifiers was also tested by selecting which classifier had the highest prediction accuracy, and this combined classifier also performed well.

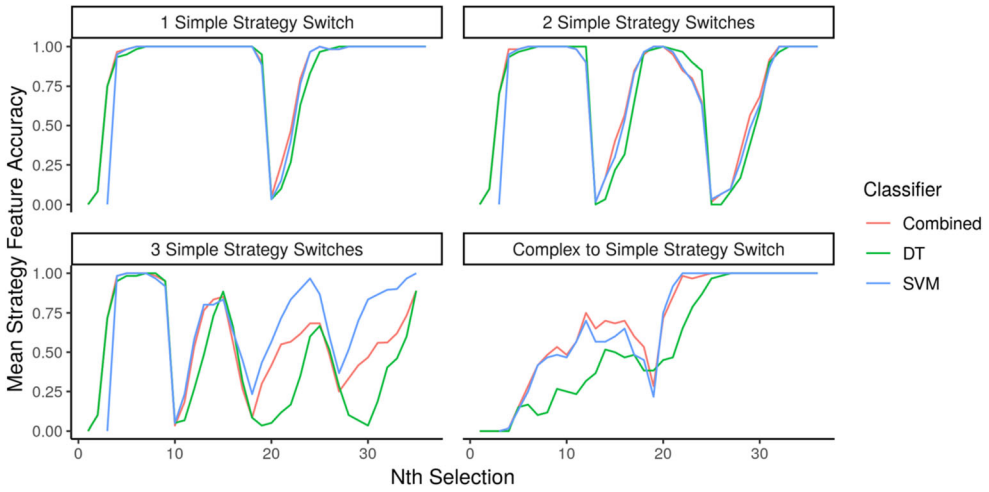

Fig. 14 Strategy feature accuracy for three different strategy switching frequencies and a complex to simply strategy switch

As shown in the top right panel of Fig. 14, all classifiers also performed well with two strategy switches. Finally, with three strategy switches in one block, a strategy change is being made after every nine selections. The bottom left panel of Fig. 14 shows that the classifiers perform best on the initial strategy, but then accuracy drops off to 50% or below on the second and third strategies. Here the SVM classifier did better than the DT, including identifying the final strategy shift. There are barely enough selections after the third strategy switch to detect the fourth strategy.

From the results of Study 1, it was expected that the classifiers would struggle to identify a two-feature strategy. The bottom right panel of Fig. 14 shows that the classifiers identified the two-feature strategy about 50–75% of the time. In the middle of the block, this model switched to a one-feature strategy, at which point the classifiers both did well. The SVM did better than the DT at detecting the complex strategy. The complex strategy was a weighted combination of the points and deadline features, which is consistent with the performance of the SVM on this strategy in Studies 1 and 2.

The final aspect of the classifiers examined is whether the grid search over possible training windows showed the expected pattern of using the maximum allowable window when the strategy had been stable for a period of time and then dropping to a smaller training window after a strategy change. This adaptive training window should allow the classifier to avoid contaminating the training data with multiple strategies by allowing it to select a window which maximizes prediction of future selections in the training window. Figure 15 shows the training window size selected by the classifiers for each of the same strategy switch scenarios for which strategy feature accuracy is shown in Fig. 14. Following a strategy switch, the training window does drop and then increase as the new strategy is used consistently for future selections. The pattern is less clear in the bottom right panel where the classifiers struggle to identify the more complex two-feature strategy initially. In particular, the SVM seemed to benefit from a longer

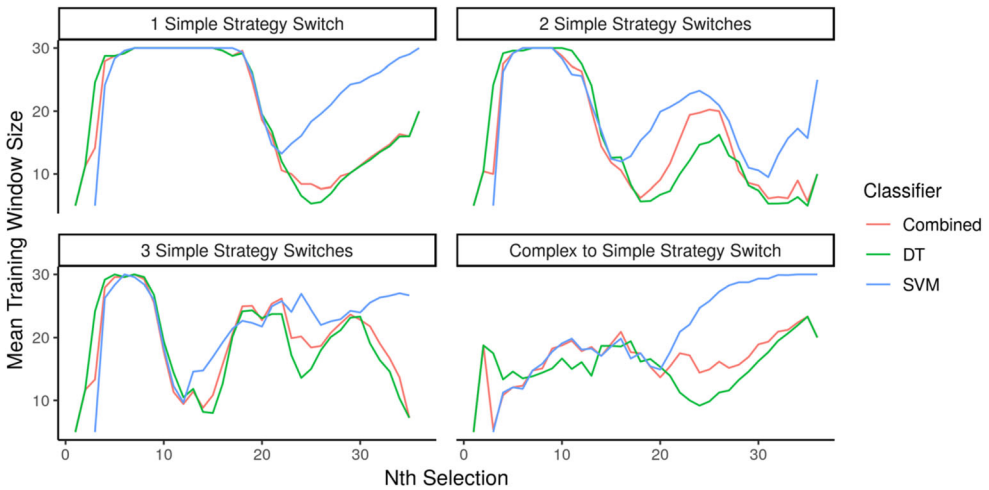

Fig. 15 Optimal training window for the different strategy switching simulations

training window after switching from the complex weighted strategy to the single feature strategy. This result may be because the single feature was a component of the weighted feature strategy and the underlying representation of the SVM.

Discussion

The results show that it is possible to use this strategy identification approach to dynamically identify strategy shifts. However, there are limits on the complexity of the strategy that can be identified with frequent strategy switches that consequently reduce the amount of relevant data for training. If people switch strategies too frequently or use more complicated selection strategies even at low levels of strategy switching, then this approach will probably not yield good data on strategies.

Therefore, it is important to consider the nature of the task when deciding whether this approach to strategy identification will yield good results. This approach will perform best when the ratio of training data to strategy switches is relatively high (e.g., above 30 selections for every strategy switch in the sADM task). Alternatively, this approach works well if strategies are constrained to be based on a single feature in which case the ratio of training data to switches can be lower (e.g., 10 in the sADM). The sADM task is a good candidate for this approach because a high degree of consistent strategy use over time is required to evaluate the effectiveness of one's strategy. Because the choice to select one object means other objects are not selected and potentially accrue a penalty, then the full impact of a strategy is only known in the task over the next few minutes. Also, time-pressured decision-making, as in the sADM, should limit the complexity of the decision-making strategy used (Oh et al., 2016). To the degree that other tasks share this property, then this approach should also be successful at identifying strategy switches in those tasks.

This approach to examining strategy switches can be used to illuminate strategies beyond classifying full blocks of trials as in Studies 1 and 2. In some blocks where the classifier reported only a partial strategy or no strategy for the entire block, it could be that participants used multiple strategies. To examine this possibility, the combined SVM/DT algorithm can be used. This algorithm is trained and makes predictions about selections over time so that it is possible to determine strategy changes if they do not occur too frequently. As in the top row of Fig. 14 for the simulated data, it is possible to identify periods of performance where the classifier returns a consistent strategy before shifting to a different one. A criterion of four or more consecutive time points with the same strategy is recommended as an indication that a stable strategy had been identified. This criterion is recommended to reduce the chance that the identification of many strategy shifts would simply be caused by noise or lack of variability in the attributes of the objects on the queue during a set of selections. For those blocks identified as having more than one strategy and a single-block classifier accuracy lower than 0.8, the strategies identified by the time-based classifier could be used provided they yield a higher classifier accuracy. Complex strategies with multiple features could show up as strategy switches using the time-based classifier, and therefore the single-block classifier would be better at identifying those. However, if a participant was really using multiple features in a single strategy over the entire block, then the non-time-based classifier would be expected to perform well. This approach for combining the single-block and time-based classifiers was adopted in the examination of human data in Study 4.

Study 4: Examining performance on human data

The prior three studies used simulations so that the strategy being used was known. However, the intent is to apply these algorithms to identify strategies from human data. Given the success of the algorithms in these parameter recovery studies, if the algorithms also show high predictive accuracy on human data, then they are likely accurately identifying the strategies that participants are using. This study illustrates the utility of these algorithms on human data.

A subset of data from a multi-session study with the sADM task was examined using the combined DT/SVM classifier and the time-based version of this same classifier. Because the strategies that participants were using are not known, a criterion for performance of the algorithm has to be selected as discussed earlier. In this study, the value recommended in Fig. 13 was used (i.e., classifier accuracy of 0.8). Predictive accuracy values above this level are likely to capture participants' strategies with values below this likely only capturing a portion of the task features participants are using.

Method

Participants There were 64 participants (aged 18–33 years, mean = 21.0 years, 44 females) who were students or staff from Mississippi State University and were paid for participation. The research was approved by the Mississippi State University Institutional Review Board.

Procedure Participants completed two sessions of the sADM task over 2 days. During the first session, participants completed a task tutorial followed by a practice block of the task in which they had to successfully sort three objects in order to proceed. They then completed eight blocks of the sADM task. In the second session conducted 2–5 days later, there was a brief set of instructions reminding them about how the task worked followed by 12 blocks of the sADM task. The primary

focus of the larger study was on how participants' strategies changed with practice on the task, including how they responded to changes in the nature of the task. The data reported here include the first eight blocks from the second session, each lasting 345 s. The final four blocks of this second session (i.e., blocks 9–12 of this session) included manipulations that promoted strategy shifts within a block, so they were not used to evaluate the performance of the algorithms.

Results

The combined DT/SVM classifier was used for this study because of its higher accuracy relative to the other classifiers in the prior studies. There were 512 blocks of sADM performance data, and a histogram of predictive accuracy is shown in Fig. 16. As a reminder, this classifier determines a single strategy for the entire block. Sixty-seven percent of the blocks had a strategy with predictive accuracy above 0.8 indicating a good likelihood of determining the strategy used by the participant in that block. For 8% of blocks, no strategy could be determined (i.e., the classifier reports that predictive accuracy is not significantly different from chance). For the remaining 25% of the blocks, the strategy determined by the classifier may only capture a portion of the features used by the participant. In addition, the mean number of features identified for each strategy was 1.29, *SD* = 0.61.

The approach for combining time-based and single-block classifiers described earlier was used to examine whether blocks for which the single-block classifier reported low accuracy might be blocks where participants switched strategies. The mean number of strategies identified per block by the time-based classifier was 1.41, *SD* = 0.82. The proportion of blocks identified as having a single strategy was 56%, which

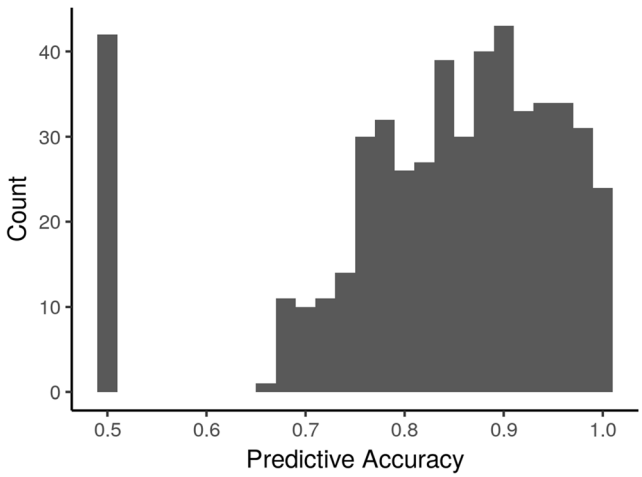

Fig. 16 Distribution of cross-validation predictive accuracy for the combined DT/SVM classifier on 512 blocks of human sADM task performance. A value of .5 (chance performance for the rank order metric used) is assigned for blocks for which no strategy could be determined

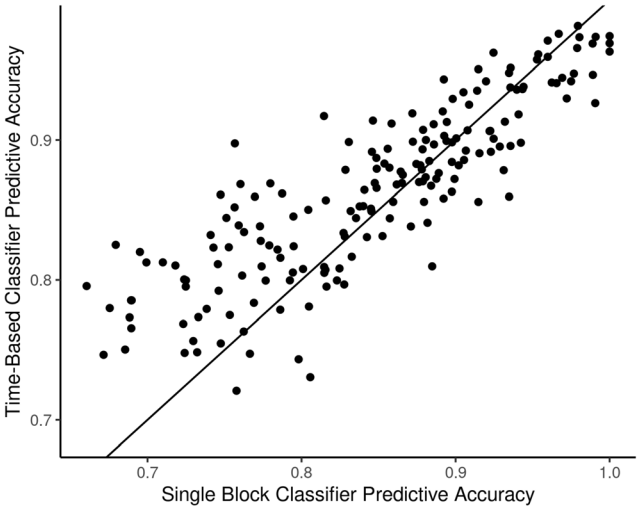

Fig. 17 A comparison of predictive accuracy for the time-based and non-time-based classifier for blocks where the time-based classifier indicated multiple strategies were used. A line with a slope of 1 is included so that points falling above the line indicate better performance by the time-based classifier

is a bit lower than the 67% of blocks estimated to have a complete strategy identified by the single-block classifier. For the blocks in which more than strategy was identified, the predictive accuracy of the time-based classifier was compared to that of the single-block classifier. As shown in Fig. 17, for these multi-strategy blocks, the time-based classifier that allows for multiple strategies generally had higher predictive accuracy as the predictive accuracy of the single-block classifier decreased below the 0.8 threshold.

For those blocks identified as having more than one strategy and a single-block classifier accuracy lower than 0.8, the predictive accuracy of the single-block classifier was replaced with that of the time-based classifier. This approach was taken because complex strategies with multiple features could show up as strategy switches using the time-based classifier. However, if a participant was really using multiple features in a single strategy over the entire block, then the single-block classifier would be expected to perform well. With that change, the percent of blocks likely to have the full strategy identified increased to 75% (compared to 67% assuming a single strategy), with another 18% likely to have a partially identified strategy (compared to 25%), and 7% having no strategy identified (compared to 8%). Being able to identify some strategy changes therefore produces a modest improvement in the ability to identify the strategies that participants are using.

It may be that many of the blocks with lower predictive accuracies come from a subset of participants because of an inconsistent use of the strategy, more satisficing behavior, or frequent shifting between strategies. Figure 18 shows that most of the blocks for which no strategy could be identified come from a small set of participants. Another subset of

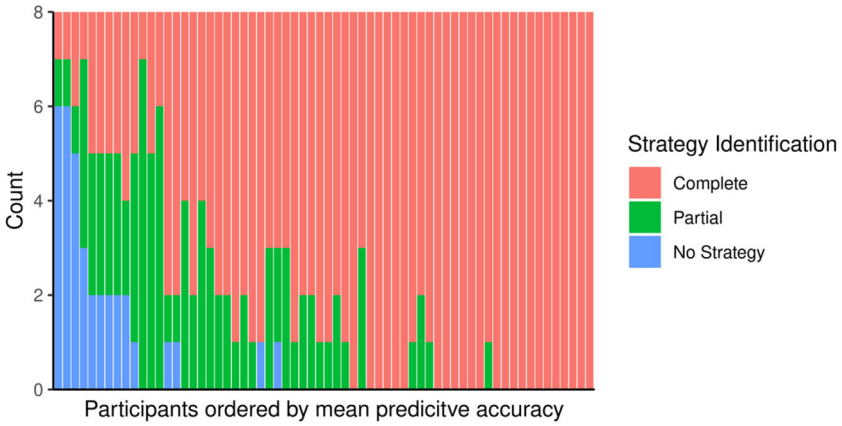

Fig. 18 An examination of how many blocks from each participant had a complete, partial, or no strategy identified by the DT/SVM classifiers. Each *vertical bar* is one participant, and they are ordered from left to right by increasing mean predictive accuracy

participants has a complete strategy identified in all blocks, and about two thirds of the participants have a complete strategy identified on 75% or more of the blocks.

The sADM is designed such that employing a consistent strategy allows one to evaluate the performance of that strategy. Because of the time that it takes to accumulate information about how a strategy does at minimizing the penalties that accrue, it should be difficult to select a high performing strategy if one switches between strategies too frequently. Therefore, consistent strategy use is both good for task performance and for the classifiers being used to identify strategies. If this line of reasoning is true, then there should be a positive correlation between predictive accuracy of the classifiers and performance in the sADM. This correlation was .32 (p = .01), supporting the hypothesis that some portion of the participants on the left half of Fig. 18 may be inconsistent strategy users (or frequent strategy shifters) who perform less well on the sADM.

Discussion

The analysis of the human data in this study indicates that the algorithms that were developed are likely to be useful in examining human strategy use in tasks such as the sADM. The fact that the sADM encourages consistent strategy use for good task performance is certainly a factor in considering the other types of tasks that it would be possible to examine with this approach. However, having some measure of strategy use determined by task behavior is a prerequisite to being able to examine a number of interesting questions about strategy development, selection, and adaptation in complex tasks.

General discussion

The results presented in these four studies show that using machine learning classifiers to identify the strategies that participants are using is promising. The first study showed that one- and two-feature strategies could be examined within a complex task with high accuracy using the amount of data that a participant would generate in 6 min of task performance. The second study showed that the classifiers handled random noise and satisficing behavior well when given twice as much task performance data. The third study showed that it would be possible to track strategy shifts across time using the same approach. The key conditions necessary for this approach to work is to have enough task performance data to identify strategies at the expected level of complexity. If participants are routinely using more than two features in their strategy, then additional task data would be required. We did not test more complex strategies in this paper because we have found no evidence that human participants in the sADM consistently use such complex strategies. For example, the mean number of features identified by the algorithms in Study 4 was 1.29 with less than 5% of identified strategies including more than two features. This is likely because of the time-pressured nature of the sADM encouraging simpler strategies, as has been found in prior research on time-pressured decision-making (Oh et al., 2016). It may also be possible to gain some insight into the strategies that participants are using by asking them to report the features they are using, but the degree to which participants can accurately report these details is itself an open question. Therefore, the approach taken here was to use simulations with known strategies to examine the degree to which this approach could recover the features used in the simulated strategies.

The general approach is to train a classifier to maximize predictive accuracy and strategy feature agreement across cross-validation folds. The results show that maximizing a weighted average of these values identifies the strategy being used in the simulated data. One potential concern with these studies is that these results will only generalize to human data to the degree that the strategies that have been simulated match the kinds of strategies that people use. The strategies

that were simulated were selected based on an analysis of heuristics reported in the multi-attribute decision-making literature (e.g., weighted additive, take the best) (Gigerenzer & Gaissmaier, 2011). Many of the strategies used were simply implemented in Lisp code inside ACT-R, so the limitations of computation that can occur in productions in ACT-R is not an issue. The ACT-R model was used because it existed for other purposes, and the constraints of this cognitive architecture did not play a role in which strategies were selected or implemented.

The results of Study 4 demonstrate that the method has both strengths and limitations in identifying human strategies. First, for most participants, the algorithm likely captured all or some features being used in their strategies. This study also found that several participants were likely using multiple selection strategies within a block of the task, even though this strategy switching was not advantageous on the sADM task. Prior work on the development of strategies in children has found significant trial-by-trial variability in strategy use even if there is a dominant strategy used in most trials (Siegler, 1991). It is likely that the approach described here would identify the dominant strategy but show lower predictive accuracy, similar to the results of Study 2, where random noise was added to the selection data. In Study 3, identification of a strategy even with the classifier that operates over trials required some number of consecutive trials on which the same strategy was used. Whether a task encourages consistent strategy use from trial to trial is therefore an additional consideration in whether this classifier-based method is appropriate.

The underlying implementation of the classifiers also played a role in the ability to accurately identify different strategies. For example, the SVM performs well on the weighted additive strategies because of its separation of selected and unselected objects using a hyperplane. Perfect adherence to a weighted additive strategy generates data that is perfectly separable by a hyperplane. The DT is more suited to strategies where participants may use more of an if-then threshold to select objects. In our results, capturing the best of both classifiers was relatively straightforward so that the results of the classifier with the highest predictive accuracy could identify the strategy.

Other types of machine learning algorithms may also be advantageous to consider in this type of approach. For example, recurrent neural networks have been shown to enable prior context or state information to inform future predictions of the network (e.g., Elman, 1990). In the sADM task, the queue from the previous selection usually is similar to the queue on the prior selection. Only one object has likely been removed and some new objects added in addition to other task-related changes that occur with the passage of time needed to sort an object. Algorithms which can track state-related information over time may be useful in these kinds of tasks, and participants may partially plan future selections as well. By contrast, the classifiers used here treat each selection as an independent event. Future work could explore more complex algorithms such as recurrent networks, but the decision to start with SVM and DT classifiers was made in order to have the ability to produce a description of the strategy as an ordered list of features with valences. Accomplishing this goal with many machine learning techniques is itself a challenging research problem related to the concept of explainable artificial intelligence (Barredo Arrieta et al., 2020).

The final point of discussion is to what degree this method can be used with other tasks. All that should be required is that the task has an outcome that is based on some strategy, and all the features that could be used in making the decision are logged along with the result of the decision. While different tasks may need to scale the data differently than was done in the case of the sADM, after this step the algorithms should be able to report the features that were used in the decision along with their valence. Therefore, the DT and SVM approach should be fairly general, with minimal work needed to function on data from another task.

There are several recommendations for use of this method. As discussed at the end of Study 2, using the combined DT/SVM classifier consistently led to results often better than either the DT or SVM classifier alone and in other cases closely approximating the performance of the individual classifiers. Using the combined classifier's reported classification accuracy at a value of 0.8 should provide reasonable confidence that the strategy's features and valences were correctly identified. For classification accuracy below this level, using the time-basedDT/SVM classifier to examine evidence for strategy switches may allow for the identification of the strategy switches as in Study 4. Hyperparameter values were explored in Studies 1–3, and our recommendation is to use the range of values documented in this paper and in the available code. The algorithm searches within this range, so it is not necessary for the researcher to specify or customize these values. The only benefit to modifying these values would be to reduce the space searched over to save processing time. However, parameters in the code parallelize the computation such that on modern multi-core/thread processors there is unlikely to be a large time savings by customizing these hyperparameter values. We have used the combined DT/SVM classifier to track strategies and have found that processing a block of sADM data takes about 20–30 s on a six-core processor. These estimates are of course dependent upon the underlying hardware.

In conclusion, this approach to identifying strategies from data generated during strategic decision-making in a complex task has been demonstrated to work reasonably well on simulated data. This method should enable

the examination of strategy selection and strategy modification behavior in tasks with a reasonably complex space of potential strategies. These developments have the potential to open new empirical opportunities to further develop theories about how people create and adapt their strategies in complex environments.

Acknowledgements This research was supported by Office of Naval Research grant N00014-17-1-2324. The authors would like to thank Kevin Barnes, Taras Krupskyy, and Daniel Roberson for assistance with data collection and testing the algorithms.

Author Note A portion of the data for Study 1 and Study 2 in this paper was reported in a conference paper (Moss et al., 2020). We have no known conflicts of interest to disclose.

References

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036–1060. https://doi.org/10.1037/0033-295X.111.4.1036

Bai, H., Jones, W. E., Moss, J., & Doane, S. M. (2014). Relating individual differences in cognitive ability and strategy consistency to interruption recovery during multitasking. *Learning and Individual Differences*, *35*, 22–33. https://doi.org/10.1016/j.lindif.2014.07.002

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., … Herrera, F. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, *58*, 82–115. https://doi.org/10.1016/j.inffus.2019.12.012

Chen, X., Bailly, G., Brumby, D. P., Oulasvirta, A., & Howes, A. (2015). The Emergence of Interactive Behavior: A Model of Rational Menu Search. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*, 4217–4226. Seoul, Republic of Korea: ACM Press. https://doi.org/10.1145/2702123.2702483

Elman, J. L. (1990). Finding structure in time. *Cognitive Science: A Multidisciplinary Journal*, *14*(2), 179–211.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd edition). O'Reilly Media.

Gigerenzer, G., & Gaissmaier, W. (2011). Heuristic decision making. *Annual Review of Psychology*, *62*, 451–482. Scopus. https://doi.org/10.1146/annurev-psych-120709-145346

Hilbig, B. E., & Moshagen, M. (2014). Generalized outcome-based strategy classification: Comparing deterministic and probabilistic choice models. *Psychonomic Bulletin & Review*, *21*(6), 1431–1443. https://doi.org/10.3758/s13423-014-0643-0

Joslyn, S., & Hunt, E. (1998). Evaluating individual differences in response to time-pressure situations. *Journal of Experimental Psychology: Applied*, *4*(1), 16–43. https://doi.org/10.1037/1076-898X.4.1.16

Klahr, D., & Dunbar, K. (1988). Dual space search during scientific reasoning. *Cognitive Science*, *12*(1), 1–48. https://doi.org/10.1207/s15516709cog1201_1

Lee, M. D. (2016). Bayesian outcome-based strategy classification. *Behavior Research Methods*, *48*(1), 29–41. https://doi.org/10.3758/s13428-014-0557-9

Lee, M. D., & Newell, B. R. (2011). Using hierarchical Bayesian methods to examine the tools of decision-making. *Judgment and Decision Making*, *6*(8), 832–842.

Lee, M. D., Gluck, K. A., & Walsh, M. M. (2019). Understanding the complexity of simple decisions: Modeling multiple behaviors and switching strategies. *Decision*, *6*(4), 335–368. https://doi.org/10.1037/dec0000105

Lovett, M. C., & Anderson, J. R. (1996). History of success and current context in problem solving: Combined influences on operator selection. *Cognitive Psychology*, *31*(2), 168–217. https://doi.org/10.1006/cogp.1996.0016

Lovett, M. C., & Schunn, C. D. (1999). Task representations, strategy variability, and base-rate neglect. *Journal of Experimental Psychology: General*, *128*(2), 107–130. https://doi.org/10.1037/0096-3445.128.2.107

Moon, J., Betts, S., & Anderson, J. R. (2013). Individual differences and workload effects on strategy adoption in a dynamic task. *Acta Psychologica*, *144*(1), 154–165. https://doi.org/10.1016/j.actpsy.2013.05.011

Moss, J., Bradshaw, G., Wong, A., Durriseau, J., Newlin, P., & Barnes, K. (2020). Tracking and improving strategy adaptivity in a complex task. In D. D. Schmorrow & C. M. Fidopiastis (Eds.), *Augmented Cognition. Human Cognition and Behavior* (pp. 416–433). Springer International Publishing. https://doi.org/10.1007/978-3-030-50439-7_29

Oh, H., Beck, J. M., Zhu, P., Sommer, M. A., Ferrari, S., & Egner, T. (2016). Satisficing in split-second decision making is characterized by strategic cue discounting. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *42*(12), 1937–1956. https://doi.org/10.1037/xlm0000284

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Prezenski, S., Brechmann, A., Wolff, S., & Russwinkel, N. (2017). A cognitive modeling approach to strategy formation in dynamic decision making. *Frontiers in Psychology*, *8*, 1335. https://doi.org/10.3389/fpsyg.2017.01335

Schunn, C. D., Lovett, M. C., & Reder, L. M. (2001). Awareness and working memory in strategy adaptivity. *Memory and Cognition*, *29*(2), 254–266. https://doi.org/10.3758/BF03194919

Schunn, C. D., McGregor, M. U., & Saner, L. D. (2005). Expertise in ill-defined problem solving domains as effective strategy use. *Memory and Cognition*, *33*(8), 1377–1387. https://doi.org/10.3758/BF03193370

Siegler, R. S. (1991). Strategy choice and strategy discovery. *Learning and Instruction*, *1*(1), 89–102. https://doi.org/10.1016/0959-4752(91)90020-9

Simon, H. A., & Lea, G. (1974). Problem solving and rule induction: A unified view. In L. W. Gregg (Ed.), *Knowledge and cognition* (pp. 105–127). Lawrence Erlbaum.

Walsh, M. M., & Gluck, K. A. (2016). Verbalization of decision strategies in multiple-cue probabilistic inference: Verbalization of decision strategies. *Journal of Behavioral Decision Making*, *29*(1), 78–91. https://doi.org/10.1002/bdm.1878

Zhang, Y., & Hornof, A. J. (2014). Understanding multitasking through parallelized strategy exploration and individualized cognitive modeling. *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems - CHI '14*, 3885–3894. Toronto, Ontario, Canada: ACM Press. https://doi.org/10.1145/2556288.2557351