What to Check Next? Bayesian Networks and Rollout-Based Lookahead for Quicker Inspections

Sebastiaan J. J. Jans S.J.J.JANS@UU.NL
Research Institute of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands

Thijs van Ommen T.VANOMMEN@UU.NL
Research Institute of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands

Ad Feelders A.J.FEELDERS@UU.NL
Research Institute of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands

Cor Veenman C.J.VEENMAN@LIACS.LEIDENUNIV.NL
Netherlands Organisation for Applied Scientific Research (TNO) and Leiden Institue of Advanced Computer Science (LIACS)
Leiden University
Leiden, The Netherlands

Editors: Gustau Camps-Valls, Manuele Leonelli and Gherardo Varando

Abstract

Regulatory inspections, such as labour inspections, take time and money. Time and money are always limited. Inspectors often work with checklists. It could save a lot of time if they could focus first on checklist items that are likely to be non-compliant with the law. We implement a method that can guide an inspector in the field by suggesting which item on the checklist they should inspect next, based on their answers to previous checklist items. To do this, we train a Bayesian Network model on filled checklist data from the Norwegian Labour Inspection Authority. We test a myopic policy that always picks the item that is most likely to be non-compliant against a policy that looks one step ahead. The lookahead policy computes for each candidate item the expected value (γ-discounted returns) of inspecting it under a fixed policy that no longer updates probabilities, also known as a rollout. The expected value under the fixed policy is cheaply computed in closed form. We find that this lookahead policy increases performance greatly over the myopic policy, halving the catching times of non-compliances from roughly 5 to 2.7 checks (out of a maximum of 14). The Bayesian network is however slightly outperformed across the board by a Binary-Relevance Naive Bayes model intended as a baseline.

Keywords: Bayesian Networks, Markov Decision Processes, Rollouts, On-Line Planning, Inspections, Checklists

© 2026 S.J.J. Jans, T. van Ommen, A. Feelders & C. Veenman.

1. Introduction

Regulatory inspections are crucial to keep a society running well, but inspection capacity is always limited. With that limited capacity, we want to achieve as much compliance as we possibly can. Since scaling up capacity is rarely possible in practice, we need to increase efficiency. Machine Learning (ML) methods can greatly help provide knowledge that steers processes towards becoming more efficient.

In risk modelling, we usually take a feature vector $\boldsymbol{x}$ of things we know about an inspectee, and use data from previous inspections to estimate a "risk score" $\hat{y}$, usually proportional to the probability of finding a non-compliance at the inspectee. These risk-based models usually model at the level of a whole inspection. The target to predict is whether a whole inspection finds a non-compliance or it does not (and possibly how serious that non-compliance is). There may be a lot to gain, however, from modelling the inspection process *at the level of individual checklist items*. Often, an inspection is essentially—or literally—the process of going through a checklist and checking whether each of the items in the list are in order. We expect that the outcomes of certain check items are correlated with the outcomes of other check items, and that we can use those correlations to find non-compliance more quickly.

In general, we are interested in finding ways to execute inspections more efficiently. We will call this the *(inspection) item planning problem*. We work on this problem together with four Dutch inspectorates. Specifically, in this work we investigate if modelling the conditional probability of non-compliance on unchecked items given previously observed (non-)compliances allows us to do that. We test a learned Bayesian network model against a naive model for the probability estimates. We also test an on-line planning method to see if looking further than one step ahead into the future improves planning. We test on a small dataset with fourteen check items, but we test only methods that generalise to more check items. On this small dataset, some quantities we approximate may be computable exactly, but since we want methods that scale, we do not pursue this.

In Section 2 we discuss related work. In Section 3 we introduce the dataset we use. In Section 4 we define the problem of predicting the probability of non-compliance and present our models to solve it. In Section 5 we state the inspection item planning problem as a Markov Decision Process, and discuss how to find a good policy. In Section 6 we analyse our results, and in Section 7 we offer conclusions.

2. Related Work

Relatively little work has been done in applying machine learning methods to risk-based inspection, especially at the level of planning an inspection per-item. Faber (2002) gives a general risk-based inspection (RBI) framework for the engineering systems domain that does include the item-planning level. Drożyner and Veith (2002) give an overview of methodologies in the same domain, and clearly emphasise focusing on the 20% of items that pose 80% of the risk. Neither of them go in-depth into the ML methods one might employ for these tasks.

Recently, the problem of planning labour inspections has been addressed by Flogard et al. (2022). Specifically they frame the problem as one of *checklist generation*. In their first work on the topic (Flogard et al., 2021) they purely use naive Bayes-like modelling in

combination with an expert-weighted similarity score to generate a (static) checklist for an inspection, based on some information about the inspectee. They see this as an instance of case-based reasoning (CBR, see for example Aamodt and Plaza, 1994). In their later work (Flogard et al., 2022), they extend this system, giving it the ability to dynamically add items to the checklist, based on the conditional probability of non-compliance of that item, given what has been observed already during the inspection.

This work is very promising, but it is also still very pioneering, and only attempts one method of prediction: CBR. Our goal is to approach the problem from a more probabilistic perspective. Also, their model only attempts to find the candidate checklist item with highest estimated probability of non-compliance (and similarity to previous cases). We suspect that there is efficiency to be gained if we look not only at whether the next item is non-compliant, but also what that may tell us about the (non-)compliance of the items we check after that. Perhaps we should not always myopically choose the most likely non-compliant item right now, but balance that with gaining information about the rest.

Note that we do not quite attempt to solve the exact same problem, since we focus more explicitly on finding non-compliances earlier within for now a small set of checklist items, while Flogard et al. (2022) use a larger dataset to also suggest checklist items that an inspector might not have thought of.

3. The Data

3.1. The Full Data

To test our methods, we need data that contains filled checklists. Flogard et al. (2021) provide such a dataset.[1] Since we use only a selection from this dataset, we will be brief in its description. It contains 59,988 inspections, in which a total of 1951 unique checklist items occur, for a total number of checks of 1,111,502. Inspectee information is limited to information about the industry in which it operates and its location. We only use the main industry area `IndustryMainAreaCode` which we call X_{isc}, though there are encodings of more detailed subdivisions too. Flogard et al. (2021) also use `MunicipalityNumber`, X_{mnr}. There is also an indicator of the region. Finally, and most importantly, each checklist item, whenever it is checked, gets a binary non-compliance label, 0 for compliant, 1 for non-compliant. It should be noted that the authors unfortunately collapsed this value into a binary variable when collecting the dataset, so "compliant" (0) might also mean "not really checked".[2] Most possible checklist items are of course not asked in most inspections, so most of the actual values are not 0 or 1 but `missing`. Because Labour Inspections inspect all sorts of entirely different organisations, it would be impossible to check the same things everywhere.

1. The full data is available for download at https://dx.doi.org/10.21227/m1t7-hg51. We use the file `LabourInspectionChecklistContent.csv`.
2. Flogard noted in personal correspondence that using more levels of the compliance variable did not achieve much in preliminary testing of his setup. It seems likely that inspectors would mostly mark things as compliant without checking if they strongly suspect it will indeed be compliant.

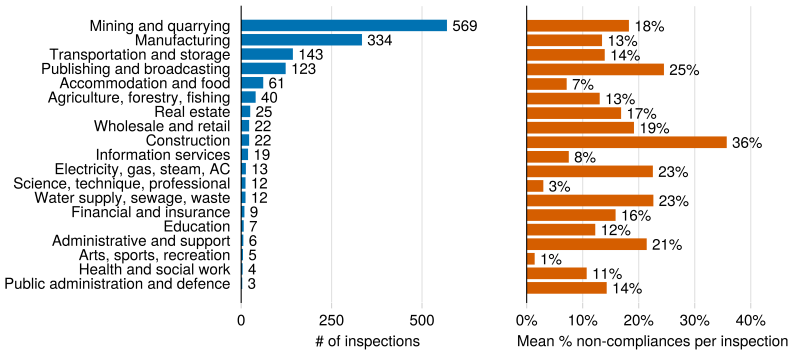


Figure 1: The number of inspections per industry (left) and average non-compliance proportion per inspection per industry (right) in the training set.

3.2. The Fully Observed Subset

In future work we would like to address the problem of incomplete data, but for now we work with a fully observed subset of the data. We obtained this subset by iteratively decreasing a level of allowed missingness, and after each decrease removing all rows and columns that were too incomplete. Of course, this opens us up to selection bias, but we do not expect this to be a problem, since our main goal is to show that this method can work on realistic data, not that it would be very good for this specific data. For the details of this scheme, we refer the interested reader to the code.[3]

We use a train-dev-test split of 60% for training all models (training set), 20% for all preliminary evaluations during development and for hyperparameter tuning (development set) and 20% for the test set, used only for the final evaluations. Inspections were assigned into sets randomly. This results in 1429, 476, and 478 inspections, respectively. The checklist items present in the data subset can be seen in Table 2 in the Appendix, along with their non-compliance percentage on the training set. See Figure 1 for the distribution of inspections over inspectee industries in the training set. Since we work with Bayesian networks, and they tend to work best when variables have limited cardinality, we cap the number of classes for the X_{isc} variable to 5 in our data subset, assigning all industries with fewer inspections than "Publishing and broadcasting" into a "rest" category.

We emphasise that a small subset like this one is still interesting to model. For example, within the Dutch labour inspectorate (NLA), there are often relatively small inspection projects based on a theme. These work with a short checklist, that is worked through completely, and therefore generate fully observed data. So there are real usecases where data like our subset is used.

4. Non-Compliance Modelling

For our methods, we need to estimate the (conditional) probability of non-compliance of a certain checklist item. In this section we describe two methods to do this. We denote the set of checklist item variables as $\boldsymbol{L}$ and we assume that they take binary values: 0 for compliant

3. The code for this paper can be found at https://github.com/Sebastiaan-Jans/inspection-policies

and 1 for non-compliant. We denote the set of additional inspectee information—such as industry, location, number of employees, etc.—by $\boldsymbol{X}$. Together they form the full vector of variables $\boldsymbol{V} = (\boldsymbol{X}, \boldsymbol{L})$. At any given timestep t during the course of an inspection, some variables are observed, and some are unobserved. We denote the set of unobserved variables at time t with $\boldsymbol{U}^{(t)} \subseteq \boldsymbol{L}$ and the observed with $\boldsymbol{O}^{(t)} \subseteq \boldsymbol{V}$. When obvious from context or otherwise irrelevant, we will drop the superscript $^{(t)}$. We will also assume that indices into subsets are automatically transformed so that for example $L_i = O_i^{(t)} = V_i$ (if $L_i \in \boldsymbol{O}^{(t)}$), even though these are technically indices into random vectors of different lengths.[4] The task of non-compliance modelling is then to find $\mathbb{P}(U_i = 1 \mid \boldsymbol{O}^{(t)} = \boldsymbol{o}^{(t)})$ for each unobserved checklist item U_i, so that we can pick the one with highest probability of non-compliance.

To be able to estimate this probability, we need models that satisfy certain requirements. Firstly, we need a model that gives us $\mathbb{P}(U_i = 1 \mid \boldsymbol{O} = \boldsymbol{o})$ for any unobserved checklist item $U_i \in \boldsymbol{L}$ and any set of observed variables $\boldsymbol{O}^{(t)} \subseteq \boldsymbol{V}$: we need *omnidirectional prediction*. The standard formulation of discriminative models, which model a fixed target given a fixed set of predictors, is not enough. Secondly, we need a model that can deal with missing data during inference (though not necessarily during training).

We think that the most natural class of model for this problem is that of graphical models. Specifically we use Bayesian networks. A Bayesian network BN is associated with a directed acyclic graph (DAG) $G = (\boldsymbol{V}, E)$ (Koller and Friedman, 2009).[5] We denote the parents of V_i in G with $\mathrm{pa}_G(V_i) = \{V_j \mid (V_j, V_i) \in E\}$. Under the Bayesian network model, the joint distribution factorises into terms that correspond to the conditional probability of each variable given its parents in the graph:

$$P_{\mathrm{BN}}(\boldsymbol{V}) = \prod_{V_i \in \boldsymbol{V}} P(V_i \mid \mathrm{pa}_G(V_i)). \tag{1}$$

This way, we only need to store a conditional probability table per variable, with size exponential only in the number of parents. The total representation is then exponential in the size of the largest parent set, as opposed to exponential in the total number of variables. Apart from efficient representation of the joint distribution, Bayesian networks are designed exactly for omnidirectional prediction (inference) and dealing with missingness of predictors during inference. They also—more so than undirected graphical models—produce a global model that is quite intuitively interpretable. Because of this they can also be wholly or partially designed or changed by experts, if they think there is something wrong with the learned model.

This leaves us with the problem of what the graph structure should be. Perhaps the simplest answer to this question is given by naive Bayes models. In a naive Bayes model, we pick a specific target, U_i, and let the edge set be $E_i = \{(U_i, V_j) \mid V_j \in \boldsymbol{V} \setminus \{U_i\}\}$. That is: we point edges from U_i to each of the other variables. The model is thus a Bayesian network with the graph $G_i = (\boldsymbol{V}, E_i)$. The factorisation, which we also call the naive Bayes assumption, is therefore given as

$$P_{\mathrm{NB}_i}(\boldsymbol{V}) = \prod_{V_j \in \boldsymbol{V}} P(V_j \mid \mathrm{pa}_{G_i}(V_j)) = P(U_i) \prod_{V_j \in \boldsymbol{V} \setminus \{U_i\}} P(V_j \mid U_i),$$

4. For an overview of used notation, see Table 1 in the Appendix.
5. For notational convenience, we identify random variables with vertices in a graphical model.

where the first step is an application of (1). This rather strong assumption that all predictors are independent of one another given the target somewhat surprisingly tends to give very good classification performance (Zhang, 2005).

We do need to train a separate naive Bayes model NB_i for each checklist item $U_i \in \boldsymbol{U}$.[6] Luckily, since training a naive Bayes model amounts to just counting cases to estimate $P(U_i)$ and each $P(V_j \mid U_i)$, this is usually still cheap enough to be doable "on the fly". If viewed as a single model for the purposes of our method, this set of naive Bayes models together forms a binary-relevance naive Bayes model (Zhang et al., 2018).

If we want a less restricted Bayesian network structure, we can instead learn it (structure and parameters) from data (Koller and Friedman, 2009). A common approach is greedy hill climbing, in which local edge additions, deletions, and reversals are greedily selected whenever they improve the network score. We use this procedure to learn the Bayesian network shown in Figure 2, using the Bayesian marginal likelihood (Bayesian Dirichlet) score for evaluating candidate structures, restricting the maximum number of parents per node to six, and placing a uniform Dirichlet prior on the conditional probability tables. Specifically, each row of every conditional probability table is assigned an independent Dirichlet$(1, \ldots, 1)$ prior (the K2 prior Cooper and Herskovits, 1992), corresponding to one pseudo-count for every possible value of the child variable under each parent configuration. We unfortunately do not have access to expert knowledge on these checklist items, so cannot validate the graph against domain expectations.

As a baseline, we use the "model" that estimates $\mathbb{P}(U_i = 1 \mid \boldsymbol{O} = \boldsymbol{o})$ by approximating it as the marginal probability $\mathbb{P}(U_i = 1)$. Though it is not sophisticated, it is of course very sensible: if scaffolding is often unsafe, while forklifts are usually in order, you do not need a complex model to tell you to check the scaffolding first.

Finally we note that, though we choose graphical, generative models because they fit the task very naturally, these are not the only type of model that may work. Standard discriminative models like logistic regression could also be made to deal with missing values, for example by coding missingness as a special value. Of course multiple models would be needed, but that is also the case for naive Bayes.

5. Finding a Good Policy

We now have models that can tell us which check item is most likely to be non-compliant. That also gives us an obvious myopic overall policy for the inspection: just inspect the item that is most likely non-compliant. However, it is not a given that that strategy optimises our goals. Our problem is similar to a treasure hunt problem (Russell and Norvig, 2020), in which we have several variables that we may choose to observe at a cost that inform us about some target variable that is valuable to know. A Bayesian network lends itself very naturally to such a problem, because we can add a utility variable to the network, and then easily calculate its expected value. The expected increase in expected utility for a variable U_i then tells us the value of (perfect) information (VOI, Howard, 1966) of that variable: how much it is worth to us to observe that variable.

6. In theory any naive Bayes model is a Bayesian network, and therefore can be used to estimate any $P(\boldsymbol{A} \mid \boldsymbol{B})$ for disjoint $\boldsymbol{A}$ and $\boldsymbol{B}$, but we should not expect these estimates to be any good.

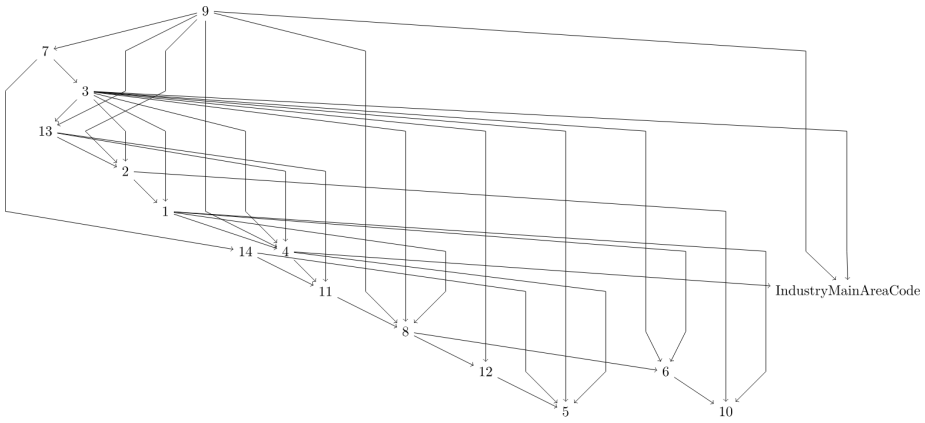


Figure 2: The learned Bayesian network. `IndustryMainAreaCode` represents the industry (encoded with levels, four for the most common industries and a fifth for the rest), and each number represents a checklist item, corresponding to the items and numbering in Table 2 in the Appendix.

In fact, we can see the myopic policy as a VOI approach where we crudely approximate the expected utility by only considering the next item. It is hard, however, to define a utility variable in a way that is generally applicable for the item planning problem. If we know exactly what inspection subdomain we are working in, we may know exactly how much finding a certain non-compliance is worth, but for this open dataset we do not have that knowledge. We also want a model that works out of the box with no expert input, and can then be *tweaked* to fit a domain even better. To achieve that, we state our general goal as follows: "Find as many non-compliances as quickly as possible."

However, it requires defining utility in a way that depends directly on the order in which we find non-compliances, which means it does not work straightforwardly as a utility node in a Bayesian network. Instead we will use the standard formalism for environments that change state depending on the action of an agent: the Markov decision process (MDP) (Bellman, 1957), well-known for its use in Reinforcement Learning (Sutton and Barto, 2018). For an overview of non-myopic adaptive VOI methods and many theoretical results we refer the reader to Golovin and Krause (2011). Our method takes a more heuristic approach, providing no theoretical guarantees, but we will show that it is very computationally cheap and we will empirically demonstrate that it works well.

In a Markov decision process, we have an environment that is in a certain state $\boldsymbol{s}^{(t)}$ at timestep t, and an agent that takes an action $a^{(t)}$ that depends on the state $s^{(t)}$. The agent then receives a reward $r^{(t)}$, and based on the previous state and the agent's action, the state transitions to the next state $s^{(t+1)}$. This can either go on until a final timestep T, or can go on forever. The agent's actions, the reward, and the state transition can all be either deterministic or stochastic. In our case we only consider stochastic state transitions and sometimes stochastic actions.

We now define the MDP we use for the inspection item planning problem.

1. Timesteps: We start at timestep 1, when nothing has been checked yet, and end at timestep $T = |\boldsymbol{L}| + 1$ when each item has been checked.

2. States encode the knowledge we currently have. That knowledge can be encoded as a state vector $\boldsymbol{s}^{(t)}$ of length $|\boldsymbol{V}|$, which contains for each check item a 0 if the item is known to be compliant ($L_i = 0$), a 1 if it is non-compliant ($L_i = 1$), and a 2 if it has not yet been checked ($L_i \in \boldsymbol{U}^{(t)}$). It also contains the values of all variables in $\boldsymbol{X}$. Then, the *state space* is $\mathcal{S} = \{0, 1, 2\}^{|\boldsymbol{L}|} \times \mathcal{X}$.

3. Taking an action in our MDP means choosing a checklist item to inspect. The *action space* is therefore at any time the set of items that have not yet been checked. To give an example (with no $\boldsymbol{X}$ features for simplicity): $\mathcal{A}(\boldsymbol{s}^{(t)}) = \mathcal{A}((0, 2, 1, 1, 2, 0)^\top) = \{L_2, L_5\} = \boldsymbol{U}^{(t)}$. The general action space is thus $\mathcal{A} = \boldsymbol{L}$.

4. The *transition distribution* $P(\boldsymbol{S}^{(t+1)} \mid \boldsymbol{S}^{(t)}, A^{(t)})$ dictates how the environment transitions from one state to the next based on the action taken by the agent. Specifically we have that $\mathbb{P}(S^{(t+1)} = \boldsymbol{s}_{U_i=c} \mid \boldsymbol{S}^{(t)} = \boldsymbol{s}, A^{(t)} = U_i) = \mathbb{P}(U_i = c \mid \boldsymbol{o}^{(t)})$, where $\boldsymbol{s}_{U_i=c}$ is the state $\boldsymbol{s}$ with U_i set to c. Obviously, the probability of transitioning to any state where any other value changes is 0. This is then where we use the non-compliance models discussed in Section 4, or of course where any other non-compliance model could be plugged in while still using this MDP architecture.

5. Our reward function is simple. We let $r^{(t)} = r(\boldsymbol{s}^{(t)}, U_i, \boldsymbol{s}^{(t+1)})$ equal 1 if U_i is found to be non-compliant, else 0. We will show later that this works well, and why it does.[7]

Given a state $\boldsymbol{s}^{(t)}$, an agent acting in an MDP takes the action $a^{(t)} = \pi(\boldsymbol{s}^{(t)})$ according to its *policy* π. When an agent starts at $t = 1$ and continues onto $t = T$, it has completed an *episode*. See Figure 3(*a*) for an example of an episode in this MDP. The sum of the rewards an agent receives when starting from timestep t we call the *return* $g^{(t)} = r^{(t)} + r^{(t+1)} + \cdots + r^{(T-1)}$. As a way to deal with infinite timesteps, it is standard to use an exponential *discounting factor* on the returns, redefining them as $g^{(t)} = r^{(t)} + \gamma r^{(t+1)} + \gamma^2 r^{(t+2)} + \ldots$, a possibly infinite sum with finite value. The returns for the example episode are then $g^{(1)} = 0 + \gamma \cdot 1 + \gamma^2 \cdot 0 = \gamma$. A nice feature of this specific discounting schema is that it is time-independent: the ratio between γ^2 and γ^3 is the same as between γ^7 and γ^8. This allows us to write nice recursions later. Incidentally, this discounting schema provides us with exactly what we need: a way to value non-compliances more when they are found earlier. This is what we will use it for. We treat γ as a hyperparameter of our method.

The value of taking an action $a^{(t)}$ in state $\boldsymbol{s}^{(t)}$ is defined by the *state-action value function* $q_\pi(\boldsymbol{s}, a)$, which can be recursively expressed in terms of $q_\pi(\boldsymbol{s}^{(t+1)}, a^{(t+1)})$. It expresses the expected returns of taking action $a^{(t)}$ and then continuing with policy π.

$$\begin{aligned} q_\pi(\boldsymbol{s}^{(t)}, a^{(t)}) &=_{\text{def}} \mathbb{E}_\pi \left[G^{(t)} \;\middle|\; \boldsymbol{S}^{(t)} = \boldsymbol{s}^{(t)}, A^{(t)} = a^{(t)} \right] \\ &= \mathbb{E}_\pi \left[R^{(t)} + \gamma G^{(t+1)} \;\middle|\; \boldsymbol{S}^{(t)} = \boldsymbol{s}, A^{(t)} = a \right] \\ &= \sum_{\boldsymbol{s}^{(t+1)}} \mathbb{P}(\boldsymbol{s}^{(t+1)} \mid \boldsymbol{s}^{(t)}, a^{(t)}) \left(r(\boldsymbol{s}^{(t)}, a^{(t)}, \boldsymbol{s}^{(t+1)}) + \gamma q_\pi(\boldsymbol{s}^{(t+1)}, \pi(\boldsymbol{s}^{(t+1)})) \right) \end{aligned} \tag{2}$$

7. We could also phrase this MDP equivalently as a Partially Observed MDP (POMDP). This allows us to avoid a special value (2) for "unobserved", but also introduces a lot of formalisms to deal with uncertain observations that we do not need here.

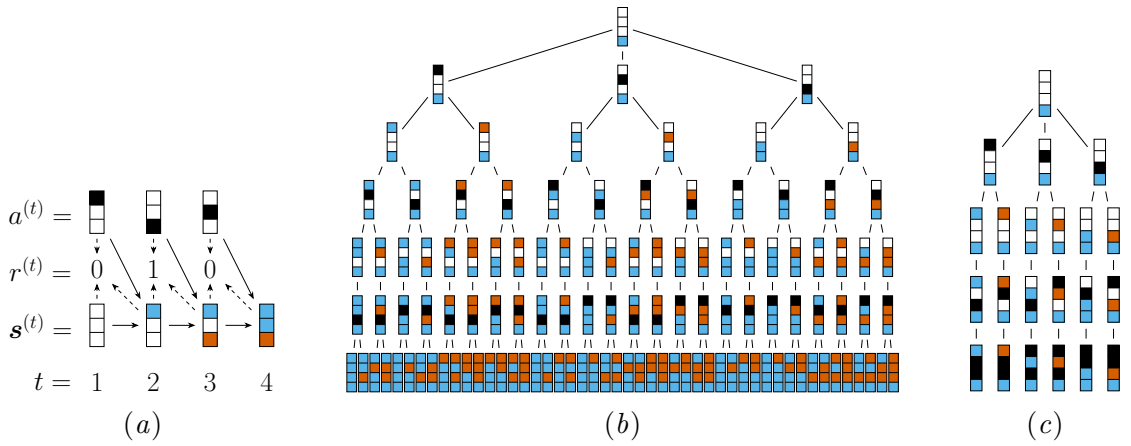


Figure 3: MDP examples, with state vectors encoded with colour. Light blue (■) represents compliant, red (■) non-compliant, and black (■) represents being chosen for inspection. No organisation information is represented. (a) Schematic episode of the MDP. (b) All possible future states after $\square\square\square\blacksquare^\top$. ($c$) The same state with fixed rollouts after the first step.

The value of a state itself is denoted $v_\pi(\boldsymbol{s}^{(t)})$. Since the policies we consider are deterministic, it is not an expectation over possible actions, but simply given by $v_\pi(\boldsymbol{s}^{(t)}) = q_\pi(\boldsymbol{s}^{(t)}, \pi(\boldsymbol{s}^{(t)}))$.

Our objective is to find an optimal policy π_*. To be precise, the optimal state-action value is defined as $q_*(\boldsymbol{s}, a) = \max_\pi q_\pi(\boldsymbol{s}, a)$, and an optimal policy selects actions according to $\pi_*(\boldsymbol{s}) = \operatorname{argmax}_a q_*(\boldsymbol{s}, a)$. If we have a model of the state transition distribution, we can in principle compute state values and state-action values exactly, by computing the recursion in (2), which we here more cleanly restate in terms of $v(\boldsymbol{s}^{(t)})$.

$$q_*(\boldsymbol{s}^{(t)}, a^{(t)}) = \sum_{\boldsymbol{s}^{(t+1)}} \mathbb{P}(\boldsymbol{s}^{(t+1)} \mid \boldsymbol{s}^{(t)}, a^{(t)})(r(\boldsymbol{s}^{(t)}, a^{(t)}, \boldsymbol{s}^{(t+1)}) + \gamma v_*(\boldsymbol{s}^{(t+1)}))$$

This takes exponential time in the number of timesteps that are left, however (Kochenderfer et al., 2022). In Figure 3(b) we illustrate the tree of possible futures from the simple state $(2, 2, 2, 0)^\top$. Even that simple state produces a lot of branches, all of which we would have to compute an expectation over. After each action in the tree (black block), we would need to run inference twice: once for a compliant outcome and once for a non-compliant outcome. While dynamic programming avoids repeatedly expanding identical subtrees (Kochenderfer et al., 2022), the number of unique states is still exponential in the number of items.

However, as noted before, in the particular case of the item planning problem, we have very reasonable and very cheap baseline strategies. Instead of computing the full expectation tree, we propose a one-step lookahead procedure. For every possible next item, we consider the possible outcomes of inspecting that item and compute the expected return after each outcome. (This requires two inference runs: one for each outcome.) Rather than recursively expanding all possible future states, we then replace the remaining expected returns v_* with the expected returns $v_{\text{fix } P(\cdot \mid \boldsymbol{o}^{(t+1)})}$ of a deterministic rollout. (See Bertsekas, 2020, for more on rollouts.) This rollout simply ranks each U_i in descending order of probability of non-compliance given the current knowledge $\boldsymbol{o}^{(t+1)}$ and checks them in that order, without

updating estimates (so no more inference runs). This we use to estimate the one-step lookahead $q_{\text{lookahead}}$ value:

$$q_{\text{lookahead}}(\boldsymbol{s}^{(t)}, a^{(t)}) = \sum_{\boldsymbol{s}^{(t+1)}} \mathbb{P}(\boldsymbol{s}^{(t+1)} \mid \boldsymbol{s}^{(t)}, a^{(t)}) v_{\text{fix } P(\cdot \mid \boldsymbol{o}^{(t+1)})}(\boldsymbol{s}^{(t+1)}).$$

We now show that this value is cheaply computable in closed form. Each $U_{(k)}$ below is ordered in descending order according to its probability $\mathbb{P}(U_{(k)} \mid \boldsymbol{S}^{(t)} = \boldsymbol{s}^{(t)} = \boldsymbol{o}^{(t)})$ and thus represents the item chosen at time k according to the fixed order.

$$\begin{aligned} v_{\text{fix } P(\cdot \mid \boldsymbol{o}^{(t)})}(\boldsymbol{s}^{(t)}) = \mathbb{E}\left[\sum_{k=t}^{T-1} \gamma^{k-t} r(\boldsymbol{s}^{(t)}, U_{(k)}, \boldsymbol{s}^{(t+1)})\right] &= \sum_{k=t}^{T-1} \gamma^{k-t}\, \mathbb{E}\left[r(\boldsymbol{s}^{(t)}, U_{(k)}, \boldsymbol{s}^{(t+1)})\right] \\ &= \sum_{k=t}^{T-1} \gamma^{k-t}\, \mathbb{P}(U_{(k)} \mid \boldsymbol{o}^{(t)}) \end{aligned} \tag{3}$$

Note the unchanging $^{(t)}$ indices in the first equality. The second equality uses the linearity of expectation, and the third uses the fact that the expectation of a binary variable is its probability. This last quantity is cheaply computed as a dot product of γ values and ranked probabilities. This approximation scheme is illustrated in Figure 3(*c*), where the expectation tree is collapsed into a single ordering of items after the first action. We get the lookahead policy that we use by taking the greedy policy w.r.t. to the lookahead state-action value. That is:

$$\pi_{\text{lookahead}}(\boldsymbol{s}^{(t)}) = \operatorname*{argmax}_{a^{(t)}} q_{\text{lookahead}}(\boldsymbol{s}^{(t)}, a^{(t)}). \tag{4}$$

We spell out the computation required for this more concretely in Algorithm 1 in the Appendix. Because we compute values as we go, not ahead of time, our method constitutes *on-line planning* (Kochenderfer et al., 2022).

Our binary reward function makes (3) very clean, though it does not actually matter which two values we choose. If we would choose to elicit utilities for finding specific non-compliances from inspectors, or costs of inspecting items, then the actual values would matter. The values also matter if we use costs for inspecting (possible negative reward) and allow early stopping. That gives a Pandora's box problem, where gathering new information may not be worth the cost, as analysed by Weitzman (1979). However, as mentioned, though our method allows for utilities and costs to be added once it is adapted for a specific inspection problem, we want our model to be usable without them, so we stick to binary rewards here.

Again, it is due to the specific dynamics of this MDP that we can produce such a rollout with closed-form expected returns, and that it gives a reasonable estimate. A more standard way of estimating value functions is by Monte Carlo Tree Search (Kochenderfer et al., 2022), which samples rollouts according to the transition distribution to estimate the expected return. This gives a good estimate of the value function, but is more expensive: we need many inference runs to get the required transition probabilities for the rollouts.

In the myopic policy, we can perform one inference run, and choose the item with highest probability of non-compliance. In a lookahead policy that takes the action with maximum

value as defined in the equations above, every time we want to select a new item, we need to run inference twice for each item: once to generate the order of items under the fixed policy if the item is compliant, and once to generate the order if it is non-compliant. This can be quite costly still, which might be a concern if we want to use our method for an inspector in the field. We could reduce this computation time, however, by only considering the most promising candidate items, since they contribute most to the expected value anyway.

6. Results

Evaluation metric To evaluate our methods, we employ a similar simulation scheme to Flogard et al. (2022). Each row in the test set represents a fully observed inspection. At the start, the agent knows nothing about the non-compliances, only the inspectee information. Then we let it decide what to check, and then we tell it the ground truth. Then it "checks" something else, until it is done checking everything. Along the way, we record at what timesteps which non-compliances were found. This procedure we repeat for each of the inspections in the test set.

Flogard et al. (2022) have to resort to estimating ground truth values, because for many cases the actual values are simply unknown because they were not actually asked in the inspection that was used as data. This leads to overconfidence in the method, though they were able to find an effect in their field study (Flogard et al., 2023) too, albeit much smaller than from the simulations. Since our data subset is fully observed, however, we do have access to the ground truth values.

Metric and results For our evaluation metric, we take the set of non-compliances, and take for each the timestep in which it was found, which we call the *catching time*. The mean of that catching time over all simulations in the test run is our target. In Figure 4 we show these means for the different policies. As comparator lines we show the random policy, which takes uniform random actions, and the marginal sorting policy, which chooses items simply in descending order of probability of non-compliance. Below that we plot the mean catching times for the lookahead policies which choose their actions according to (4), one line using the BN model and the other using the binary relevance Naive Bayes model for conditional probability updates. We tested several values for the discounting rate γ, the hyperparameter that determines how much future rewards are weighted. The discounting rates were chosen through experimentation on the development set. The values around $\gamma = 0.9$ were chosen because that is where peak performance occurred on the development set, and lower values were chosen to illustrate that performance drops off sharply with γ below 0.75. For $\gamma = 0$ we in fact ran the myopic policy, because it is then unnecessary to do another inference run. All of these models clearly outperform the random policy. It is interesting to note, however, that the myopic policies with NB or BN model do not strongly outperform the simplest marginal probability ranking method.

7. Conclusion and Future Work

We showed in this work that Bayesian networks and naive Bayes models can help shorten inspections. It seems that naive Bayes may be the better option, since it is cheaper and seems to slightly outperform the Bayesian network. While myopic models that take previous

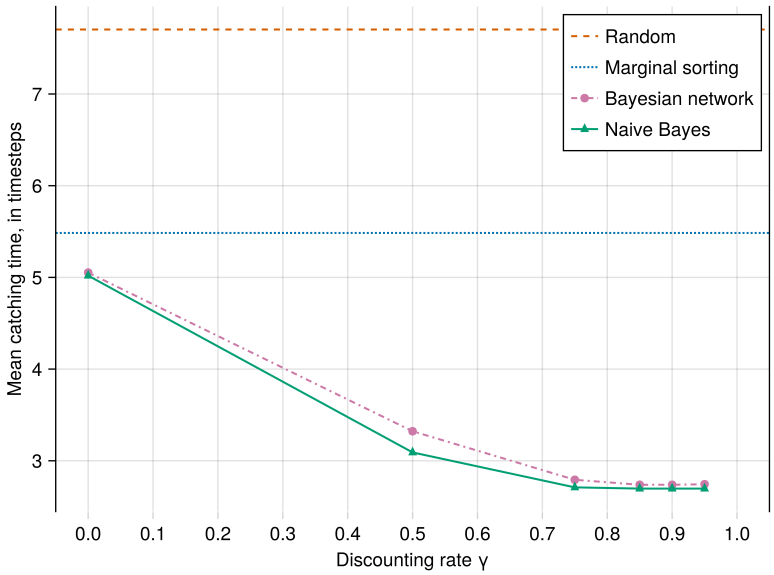


Figure 4: Mean catching time (lower is better) of non-compliances on the test set for different discounting rates γ. (With $\gamma = 0$ the policy is fully myopic.)

checklist answers into account show some improvement over simply ranking all items by their marginal probability, they are far outperformed by calculating expected returns with a fixed policy/rollout, looking a step ahead. Whether this is worth the computational cost is a practical question that must be answered by an inspectorate.

Future avenues for research might include dealing more fully with unobserved data, like we see in the full dataset. A naive Bayes model can easily deal with this, and a Bayesian network model might be made to handle it too. A major issue, however, is that it is hard to validate results in this setting, because many ground truth values are missing. It would also be interesting to work with data that includes more checklist items. This would also make the decrease in catching times clearer. To keep computation feasible, we could consider not taking all candidate checklist items into account for the expected return calculation.

Acknowledgments

This research was done in the context of the AI4Oversight lab, which is sponsored by four Dutch inspectorates—the Netherlands Labour Authority (NLA), the Inspectorate of Education (IvhO), the Human Environment and Transport Inspectorate (ILT) and the Netherlands Food and Consumer Product Safety Authority (NVWA)—and the Netherlands Organisation for Applied Scientific Research (TNO).

This work was produced by humans, at points aided by Large Language Models. We used ChatGPT and Claude for sparring about ideas, checking passages, helping with producing (Ti*k*Z) images. Most code was written by the authors, but some was left to LLMs and only checked by humans. All final text was written by the authors.

References

Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59, 1994.

Richard Bellman. A Markovian decision process. *Journal of mathematics and mechanics*, 6(5):679–684, 1957.

Dimitri P Bertsekas. *Rollout, policy iteration, and distributed reinforcement learning*. Athena Scientific Belmont, MA, 2020.

Gregory F Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992. doi: 10.1007/BF00994110.

Przemysław Drożyner and Eric Veith. Risk based inspection methodology overview. *Diagnostyka*, 27, 2002.

Michael H. Faber. Risk-based inspection: An introduction. *Structural Engineering International*, 12(3):186–195, 2002. doi: 10.2749/101686602777965388.

Eirik Lund Flogard, Ole Jakob Mengshoel, and Kerstin Bach. Bayesian feature construction for case-based reasoning: Generating good checklists. In *International Conference on Case-Based Reasoning*, pages 94–109, 2021.

Eirik Lund Flogard, Ole Jakob Menghshoel, and Kerstin Bach. Creating dynamic checklists via Bayesian case-based reasoning: Towards decent working conditions for all. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5108–5114, 2022. doi: 10.24963/ijcai.2022/709.

Eirik Lund Flogard, Ole Jakob Menghshoel, Ole Magnus Theisen, and Kerstin Bach. Creating explainable dynamic checklists via machine learning to ensure decent working environment for all: A field study with labour inspections. In *Prestigious Applications of Intelligent Systems (PAIS)*, pages 3218–3225, 2023.

Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011. doi: 10.1613/jair.3278.

Ronald A. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966. doi: 10.1109/TSSC.1966.300074.

Mykel J Kochenderfer, Tim A Wheeler, and Kyle H Wray. *Algorithms for decision making*. MIT press, 2022.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2020.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

Martin Weitzman. Optimal search for the best alternative. *Econometrica*, 47(3):641–654, 1979. doi: 10.2307/1910412.

Harry Zhang. Exploring conditions for the optimality of naive Bayes. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(02):183–198, 2005.

Min-Ling Zhang, Yu-Kun Li, Xu-Ying Liu, and Xin Geng. Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, 12(2):191–202, 2018. doi: 10.1007/s11704-017-7031-7.

Appendix A. Additional Details

In Table 1 we give an overview of the notation we use. In Table 2 we give the English translations of the checklist items in the dataset (original in Norwegian). In Algorithm 1 we detail the exact way in which our method chooses its action and computes the necessary quantities.

Table 1: Notation overview

Notation	Meaning
$\boldsymbol{V}$	A random vector. $\boldsymbol{V} = (V_1, \ldots, V_n)^\top$
V_i	The i-th random variable of $\boldsymbol{V}$.
$\mathcal{V}$	The sample space of $\boldsymbol{V}$.
$\boldsymbol{v}$	A configuration of values, $\boldsymbol{v} \in \mathcal{V}$ and $\boldsymbol{v} = (v_1, \ldots, v_n)^\top$.
$P(\boldsymbol{V})$	(Joint) probability distribution of $\boldsymbol{V}$.
$\mathbb{P}(\boldsymbol{V} = \boldsymbol{v})$ or $\mathbb{P}(\boldsymbol{v})$	Probability measure.
$P(\boldsymbol{L} \mid \boldsymbol{X})$	Conditional probability distribution.
$\mathbb{P}(\boldsymbol{L} = \boldsymbol{l} \mid \boldsymbol{X} = \boldsymbol{x})$ or $\mathbb{P}(\boldsymbol{l} \mid \boldsymbol{x})$	Conditional probability measure.
$\mathbb{E}[\boldsymbol{V}]$	Expectation. $\mathbb{E}[\boldsymbol{V}] = (\mathbb{E}[V_1], \ldots, \mathbb{E}[V_n])^\top$
$\mathbb{E}[\boldsymbol{L} \mid \boldsymbol{X}]$	Conditional expectation.
$(\boldsymbol{X}, \boldsymbol{L})$	Concatenation of random vectors.
$P_{\mathcal{M}}$	Distribution defined by model $\mathcal{M}$.

Function `v_fixed`($\boldsymbol{p}_{\text{nc}}$)
  `sort`($\boldsymbol{p}_{\text{nc}}$, `descending` = `true`)
  $n \leftarrow$ `length`($\boldsymbol{p}_{\text{nc}}$)
  return $\sum_{i=1}^{n} \gamma^{i-1} \boldsymbol{p}_{\text{nc}}[i] \quad (= \boldsymbol{p}_{\text{nc}}[1] + \gamma \boldsymbol{p}_{\text{nc}}[2] + \gamma^2 \boldsymbol{p}_{\text{nc}}[3] + \cdots + \gamma^{n-1} \boldsymbol{p}_{\text{nc}}[n])$

Function `q_lookahead`($\boldsymbol{s}$, U_i, p_{nc})
  foreach $c \in \{0, 1\}$ **do**
    $\boldsymbol{s}_{U_i=c} \leftarrow \boldsymbol{s}$ with U_i set to c
    $\boldsymbol{p}_{\text{nc}|U_i=c} \leftarrow$ `infer`($\mathcal{M}$, $\boldsymbol{s}_{U_i=c}$)
    $v_{\text{fix } P(\cdot \mid \boldsymbol{o}^{(t+1)})}(\boldsymbol{s}_{U_i=c}) \leftarrow$ `v_fixed`($\boldsymbol{p}_{\text{nc}|U_i=c}$)
  end
  return $(1 - p_{\text{nc}}) \cdot v_{\text{fix } P(\cdot \mid \boldsymbol{o}^{(t+1)})}(\boldsymbol{s}_{U_i=0}) + p_{\text{nc}} \cdot v_{\text{fix } P(\cdot \mid \boldsymbol{o}^{(t+1)})}(\boldsymbol{s}_{U_i=1})$

Function `greedy_next_item`($\boldsymbol{s}^{(t)}$)
  $\boldsymbol{p}_{\text{nc}} \leftarrow$ `infer`($\mathcal{M}, \boldsymbol{s}^{(t)}$)
  foreach $(U_i, p_{\text{nc}}) \in \boldsymbol{U}^{(t)} \times \boldsymbol{p}_{\text{nc}}$ **do**
    $q_{\text{lookahead}}(\boldsymbol{s}^{(t)}, U_i) \leftarrow$ `q_lookahead`($\boldsymbol{s}^{(t)}, U_i, p_{\text{nc}}$)
  end
  return $\text{argmax}_{U_i}(q_{\text{lookahead}}(U_i))$

Algorithm 1: Concrete illustration of how our method greedily selects an action (`greedy_next_item`) using the state-action value computed by `q_lookahead`, which in turn uses the fixed-rollout approximation computed in `v_fixed`. We use the obvious definitions for `sort` and `length`, and `infer`($\mathcal{M}, \boldsymbol{s}$) runs model inference and outputs a vector of length $U^{(t)}$ which contains the probability of non-compliance for each yet unobserved item.

Table 2: Checklist items in data selection, with the percentage of non-compliance in the training set

#	**Check item text**	**%NC**
1	Has the employer implemented the necessary measures and/or prepared a plan describing measures to remove or reduce hazards and problems at work?	33.4%
2	Does the employer ensure that the safety representative receives the necessary training?	30.4%
3	Has the employer implemented a routine for how deviations (undesirable events) are to be reported and followed up?	29.7%
4	Has the company chosen a safety representative?	23.6%
5	Has the employer ensured that the employee appointed to lead others has the necessary competence so that considerations of safety and health are taken care of within the manager's area of responsibility?	17.6%
6	Does the employer ensure that a written employment agreement is entered into with the employees?	17.1%
7	Is the business affiliated with an approved occupational health service?	16.9%
8	Has the employer undergone training in health, environment and safety work?	16.2%
9	Does the employer ensure that a plan has been prepared for the occupational health service's assistance in the company?	16.0%
10	Does the employer have a continuous overview of how much the individual employee works?	11.4%
11	Has the employer ensured that the employment agreements are in line with the minimum requirements set out in the Working Environment Act §14-6?	6.2%
12	Does the employer ensure that the safety representative is consulted in the company's work in planning and implementing measures that are important for the working environment?	6.1%
13	Has the employer mapped the dangers and problems the employees may be exposed to in the company and, on this basis, assessed the risk of injury or danger to employees' health and safety?	5.3%
14	Has the employer ensured that employees have been made aware of accident and health hazards associated with the work, and that they have received the training, exercise, and instruction necessary to ensure workplace safety?	1.7%