Noname manuscript No.
(will be inserted by the editor)

Minimizing Data Consumption with Sequential Online Feature Selection

Thomas Rückstieß · Christian Osendorfer · Patrick van der Smagt

Received: date / Accepted: date

Abstract In most real-world information processing problems, data is not a free resource. Its acquisition is often expensive and time-consuming. We investigate how such cost factors can be included in supervised classification tasks by deriving classification as a sequential decision process and making it accessible to Reinforcement Learning. Depending on previously selected features and the internal belief of the classifier, a next feature is chosen by a sequential online feature selection that learns which features are most informative at each time step. Experiments on toy datasets and a handwritten digits classification task show significant reduction in required data for correct classification, while a medical diabetes prediction task illustrates variable feature cost minimization as a further property of our algorithm.

Keywords reinforcement learning · feature selection · classification

1 Introduction

With the increasing computerization of society, the amount of stored data appears to grow rapidly, without the corresponding growth in underlying information. According to various studies[1], recent years showed an annual 40–60% increase of commercial storage needs and a 40+-fold increase is expected in the next decade. In other words, the *redundancy* of data continuously increases. It has been commonly understood that data redundancy negatively impacts the performance of

Thomas Rückstieß and Christian Osendorfer
Institute of Computer Science VI
Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany
E-mail: ruecksti@in.tum.de, osendorf@in.tum.de

Patrick van der Smagt
DLR / Institute of Robotics and Mechatronics
P.O.Box 1116, 82230 Wessling, Germany
E-mail: smagt@dlr.de

[1] E.g., Gartner's survey at `http://www.gartner.com/it/page.jsp?id=1460213`.

classification methods. For that reason, it is good practice to reduce data redundancy in a pre-processing *feature selection* (FS) step, using methods such as dimensionality reduction, PCA, etc. to extract few meaningful features that can explain the data.

Going beyond traditional FS methods, in this paper we demonstrate an approach to select features in sequence, making the decision which feature to select next (a) online, during classification and (b) dependent on previously selected features and the current internal state of the classifier. In particular, we will reduce *data consumption* by embedding reinforcement learning (RL) into classification tasks leading to our sequential online feature selection (SOFS). The question we address in this paper is: "*Where do I have to look next, in order to keep data consumption and expenses low while maintaining good classification results?*"

Feature selection with RL has been previously addressed [**?**], yet the novelty of our approach lies in its sequential decision process. Our work is based on and inspired by existing research, combining aspects of online FS [**?**,**?**] and attentional control policy learning [**?**,**?**,**?**]. A similar concept, Online Streaming FS [**?**], has features streaming in one at a time, where the control mechanism can accept or reject the feature. While we adopt the idea of sequential online feature selection, our scenario differs in that it allows access to all features with the subgoal of minimizing data consumption.

Also closely related to our work is [**?**], classifying inputs in a sequential manner based on approximate policy iteration. Their work uses the features directly rather than the classifier belief, leading to a more integrated approach where both classification and feature selection is done by a single component.

Related work further includes [**?**] which uses RL for feature learning in object tracking dependent on visual context, and [**?**], where RL is used to create an ordered list of image segments based on their importance for a face recognition task. However, their decision process is not dependent on the internal state of the classifier, which brings their method closer to conventional FS.

Despite the similar name, our approach differs from popular sequential feature selection methods like "Sequential Forward Floating Selection" (SFFS, e.g., [**?**]), as they still choose features prior to classification and do not allow selection of different features for individual data samples.

Our framework is mapped out in Section 2. After introducing the general idea, we formally define sequential classifiers and rephrase the problem as a Partially Observable Markov Decision Process (POMDP). In addition, a novel action selection mechanism without replacement is introduced. Section 3 then demonstrates our approach, first on two artificially created toy examples, then on real-world problems, both with redundant (handwritten digit classification) and costly (diabetes classification) data and discusses the results.

2 Framework

2.1 General Idea

In Machine learning, solving a classification problem means to map an input x to a label $c \in \mathcal{C}$. Classification algorithms are trained on labelled training samples

$I = \{(x^1, c^1), \ldots, (x^n, c^n)\}$, while the quality of such a learned algorithm is determined by the generalization error on a separate test set. We regard features as disjunct portions (scalars or vectors) of the input pattern x, with feature labels $f_i \in F$ and feature values $f_i(x)$ for feature f_i. One key ingredient for good classification results is feature selection (also called *feature subset selection*): filtering out irrelevant, noisy, misleading or redundant features. FS is therefore a combinatorial optimization problem that tries to identify those features which will minimize the generalization error. In particular, FS tries to reduce the amount of useless or redundant data to process.

We wanted to take this concept even further and focus on minimizing *data consumption*, as outlined in the introduction. For this purpose, however, FS is not ideal. Firstly, the FS process on its own commonly assumes free access to the full dataset. But more significantly, FS determines for *any* input the *same subset* of features that should be used for a subsequent classification. We argue that this limitation is not only unnecessary, but in fact disadvantageous in terms of minimizing data consumption.

By turning classification into a sequential decision process, we can significantly reduce the amount of data to process. In that case, FS and classification become a closely intertwined process: deciding which feature to select next depends on the previously selected features and the behaviour of the classifier on them.

To demonstrate that RL can be used to select features in an online, sequential manner, we will take a fully trained classifier as an environment for an RL agent, which learns which feature to access next, receiving reward on successful classification of the partially uncovered input pattern. While training the classifier requires access to the full training dataset, subsequent classifications (e.g., on a verification dataset or on unseen data) will access fewer features (consume less data) and therefore incur reduced feature costs. It is further possible to train the classifier and SOFS simultaneously for even greater cost reduction. However, this is not within the scope of this paper and will be addressed in a future publication.

2.2 Additional Notation

For the next sections, we additionally require the following notation: ordered sequences are denoted by $(\cdot)$, unordered sets are denoted by $\{\cdot\}$, appending an element e to a sequence s is written as $s \circ e$. Related to power sets, we define a *power sequence* powerseq(M) of a set M to be the set of all ordered permutations of all elements of the power set of M, including the empty sequence $()$. As an example, for $M = \{1, 2\}$, the resulting powerseq$(M) = \{(), (1), (2), (1, 2), (2, 1)\}$. During an episode, the feature history $h_t \in$ powerseq(F) is the sequence of all previously selected features in an episode up to and including the current feature at time t. Costs associated with accessing a feature f are represented as negative scalars $r_f^- \in \mathbb{R}, r_f^- < 0$. We further introduce a non-negative global reward $r^+ \in \mathbb{R}, r^+ \geq 0$ for correctly classifying an input. A classifier in general is denoted by K, and sequential classifiers (defined in Section 2.3) are written as $\widetilde{K}$.

2.3 Sequential Classification

We define a *sequential* classifier $\widetilde{K}$ to be a functional mapping from the power sequence of feature values to a set of classes:

$$\widetilde{K} : \text{powerseq}\left(\{f(x)\}_{f\in F}\right) \to \mathcal{C} \tag{1}$$

Our framework assumes that feature values are passed to the classifier $\widetilde{K}$ one at a time, therefore $\widetilde{K}$ requires some sort of memory. Recurrent Neural Networks (RNN) [**?**], for instance, are known to have implicit memory that can store information about inputs seen in the past. If the classifier does not possess such a memory, it can be provided explicitly: at timestep t, instead of presenting only the t-th feature value $f_t(x)$ to the classifier, the whole history $(f_1(x), \ldots, f_t(x))$ up to time t is presented instead.

As it turns out, the above approach of providing explicit memory can also be used to turn any classifier which can handle *missing values* [**?**] into a sequential classifier. For a given input x and a set F_1 of selected features, $F_1 \subseteq F$, the values of the features not chosen, i.e., $F \backslash F_1$, are defined as *missing*, which we will denote as ϕ. Each episode starts with a vector of only missing values $(\phi, \phi, \ldots)$, where ϕ can be the mean over all values in the dataset, or simply consist of all zeros. More sophisticated ways of dealing with missing values based on imputation methods [**?**] can be implemented accordingly. At each time step, the current feature gradually uncovers the original pattern x more. As an example, assuming scalar features f_1, f_4 and f_6 were selected from an input pattern $x \in \mathbb{R}^6$, the input to the classifier K would then be: $(f_1(x), \phi, \phi, f_4(x), \phi, f_6(x))$. This method allows us to use existing, pretrained non-sequential classifiers in a sequential manner. Note that the classifiers will remain unchanged and only act as an environment in which the SOFS agent learns. We therefore do not measure the performance of the classifiers but rather the number of features necessary until correct classification was achieved.

2.4 Classification as POMDP

We will now reformulate classification as a Partially Observable Markov Decision Process[2] (POMDP) [**?**], making the problem sequential and thus accessible to Reinforcement Learning algorithms.

To map the original problem of classification under the objective to minimize data consumption to a POMDP, we define each of the elements of the 6-tuple $(S, A, O, \mathcal{P}, \Omega, \mathcal{R})$, which describes a POMDP, as follows: the state $s \in S$ at timestep t comprises the current input x, the classifier $\widetilde{K}$, and the previous feature history h_{t-1}, so that $s_t = (x, \widetilde{K}, h_{t-1})$. This triple suffices to fully describe the decision process at any point in time. Actions $a_t \in A$ are chosen from the set of features $F \backslash h_{t-1}$, i.e., previously chosen features are not available. Section 2.5 describes how this can be implemented.

[2] A *partially observable* MDP is a MDP with limited access to its states, i.e., the agent does not receive the full state information but only an incomplete observation based on the current state.

The observation is represented by the classifier's internal belief of the class after seeing the values of all features in h_{t-1}, written as $o_t = b(x, \widetilde{K}, h_{t-1}) = b(s_t)$. Most classifiers base their class decision on some internal belief state. A Feed Forward Network (FFN) for example often uses a softmax output representation, returning a probability p_i in $[0, 1]$ for each of the classes with $\sum_{i=1}^{|\mathcal{C}|} p_i = 1$. And if this is not the case (e.g., for purely discriminative functions like a Support Vector Machine), a straightforward belief representation of the current class is a k-dimensional vector with a 1-of-k coding. In the experiments section, we will demonstrate examples with FFN, RNN and Naive Bayes classifiers. Each of these architectures allows us to use the aforementioned softmax belief over the classes as belief state for the POMDP. The probabilities p_i for each class serve as an observation to the agent:

$$o_t = b(x, \widetilde{K}, h_{t-1}) = (p_1, p_2, \ldots, p_{|\mathcal{C}|}) \tag{2}$$

Assuming a fixed x and a deterministic, pretrained classifier $\widetilde{K}$, the state and observation transition probabilities $\mathcal{P}$ and Ω collapse and can be described by a deterministic transition function T, resulting in the next state and observation:

$$s_{t+1} = T_x(s_t, a_t) = (x, \widetilde{K}, h_{t-1} \circ a_t) \tag{3}$$

$$o_{t+1} = b\left(s_{t+1}\right) \tag{4}$$

Lastly, the reward function $\mathcal{R}$ returns the reward r_t at timestep t for transitioning from state s_t to s_{t+1} with action a_t. Given c as the correct class label, it is defined as:

$$r_t = \begin{cases} r^+ + r^-_{a_t} & \text{if } \widetilde{K}\left((h_\tau(x))_{0<\tau\leq t}\right) = c \\ r^-_{a_t} & \text{else} \end{cases} \tag{5}$$

2.5 Action Selection without Replacement

In this specific task we must ensure that an action (a feature) is only chosen at most once per episode, i.e., the set of available actions at each given decision step is dependent on the history h_t of all previously selected actions in an episode. Note that this does not violate the Markov assumption of the underlying MDP, because no information about available actions flows back into the state and therefore the decision does not depend on the feature history.

Value-based RL offers an elegant solution to this problem. By manually changing all action-values $Q(o, a_t)$ to $-\infty$ after choosing action a_t, we can guarantee that all actions not previously chosen in the current episode will have a larger value and be preferred over a_t. A compatible exploration strategy for this action selection without replacement is Boltzmann exploration. Here, the probability of choosing an action is proportional to its value under the given observation:

$$p(a_t|o_t) = \frac{e^{Q(o_t,a_t)/\tau}}{\sum_a e^{Q(o_t,a)/\tau}}, \tag{6}$$

where τ is a temperature parameter that is slowly reduced during learning for greedier selection towards the end. Thus, when selecting action a_{t+1}, all actions in h_t have a probability of $e^{-\infty} = 0$ of being chosen again. At the end of an episode, the original Q-values are restored.

Algorithm 1 Sequential Online Feature Selection (SOFS)

Require: labelled inputs I, agent A, sequential classifier $\widetilde{K}$
1: **repeat**
2: choose $(x, c) \in I$ randomly
3: $h_0 \leftarrow (\phi)$
4: $o_1 \leftarrow b(x, \widetilde{K}, h_0)$
5: **for** $t = 1$ to $|F|$ **do**
6: $a_t \leftarrow \mathrm{A}(o_t)$
7: $h_t \leftarrow h_{t-1} \circ a_t$
8: $o_{t+1} \leftarrow b(x, \widetilde{K}, h_t)$
9: **if** $\widetilde{K}\left((h_\tau(x))_{0<\tau\leq t}\right) = c$ **then**
10: $r_t \leftarrow (r^+ + r^-_{a_t})$
11: **break**
12: **else**
13: $r_t \leftarrow r^-_{a_t}$
14: **end if**
15: **end for**
16: train A with $(o_1, a_1, r_1, \ldots, r_t, o_{t+1})$
17: **until** convergence

2.6 Solving the POMDP

Having defined the original task of classification with minimal data consumption as a POMDP and solved the problem of action selection without replacement, we can revert to existing solutions for this class of problems. Since the transition function is unknown to the agent, it needs to learn from experience, and a second complication is the continuous observation space. For regular MDPs, a method well-suited to tackle both of these issues is Fitted Q-Iteration (FQI) [**?**]. The sequential classifier $\widetilde{K}$ then takes care of the *PO* part of the POMDP, yielding a static belief over the sequential input stream.

FQI uses a batch-trained function approximator (FA) as action-value function. Various types of non-linear function approximators have been successfully used with FQI, e.g., Neural Networks [**?**], CMACs [**?**], Gaussian Processes [**?**], Advantage Weighted Regression [**?**], and others [**?**]. In this paper, we will use Locally Weighted Projection Regression (LWPR) [**?**] as the value function approximator of choice, as it is a fast robust online method that can handle large amounts of data.

The details of the algorithm are presented in Listing 1. The history is always initialized with the missing value ϕ (line 3). This gives the system the chance to pick the first feature before seeing any real data. The SOFS agent is trained after every episode (line 16), which ends either with correct classification (line 9–11) or when the whole input pattern was uncovered (line 15), i.e., all features were accessed. Training is continued until the algorithm converges, i.e., the average episodic return no longer significantly improves.

3 Experiments and Discussion

We evaluate the proposed method on four different datasets to demonstrate and point out certain properties of SOFS: two artificial toy examples, the MNIST handwritten digits classification task, and a medical dataset for diabetes predic-

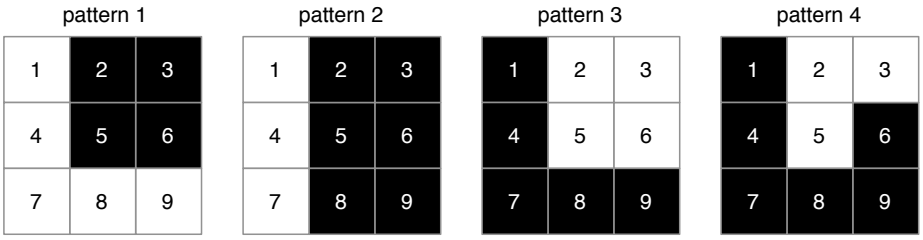

Fig. 1 Artificial toy data to investigate whether SOFS bases its decision on the current state or simply chooses informative features (like regular FS) independent of the state. Pattern 1 and 2 can be distinguished with features 8 or 9, while pattern 3 and 4 can be distinguished with feature 6.

tion. Each experiment was repeated 25 times, the plots for MNIST and the diabetes task show single runs (gray) and the mean value over all runs (black).

3.1 Toy Example I—Shapes

This toy dataset was inspired by the MNIST handwritten digits set (Section 3.3) but is much simpler, has a lower dimension and only 4 different patterns, illustrated in Figure 1. It was chosen to get an insight into the decision-making process of a trained SOFS agent, which a large dataset like MNIST cannot provide that easily.

Each pattern consists of 3×3 pixels, and each pixel was considered a feature. We used artificially created training data (1000 samples, each randomly chosen from the four patterns). Before training the SOFS agent, we pre-trained a FFN classifier with a 9-20-4 architecture, sigmoid activation functions in the hidden layer, and softmax activation in the output layer. Training was conducted sequentially, using the explicit memory approach from Section 2.3. The class targets used 1-of-n encoding, training was conducted over the full dataset for 30 epochs with a learning rate $\alpha = 0.1$.

The FQI agent was then trained over 600 episodes according to Section 2.6. To evaluate the learned behavior, exploration was deactivated, rendering the whole process deterministic. The system was then presented with all four input patterns. Figure 2 illustrates its response for each case and the decision process during an episode. Since only four patterns were used without any noise, the system quickly converged to a perfect solution, always classifying the correct pattern after looking at 2 features at most.

3.2 Toy Example II—Cube

In this second toy example, we created an artificial dataset with an arbitrary number of features F, which can be set upon creation of the data. The idea is to have some useful information hidden in each data point while most of the features are non-informative random values. The position of the useful features are dependent on the class label, however.

This is how we created the cube dataset: Three of the features indicate x-, y-, and z-coordinates in a three-dimensional space. Each of the coordinates was

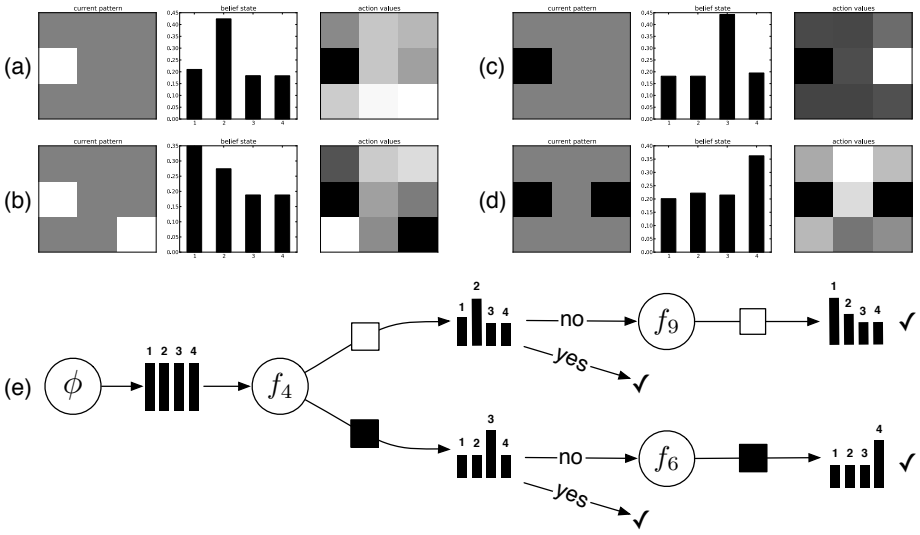

Fig. 2 Decision process of the SOFS agent after training. (a)–(d) each show the already uncovered features in the explicit memory (left), the belief state histogram (middle) and the action value table for selecting the next feature (right, white indicates high values). (e) shows the decision process graphically. Initially the agent always sees the *missing* value and chooses to look at feature 4 first. If the feature is white, the classifier favours class 2, and SOFS proposes to select feature 9, should it be wrong (a). After looking at feature 9, the classifier then favours class 1 (b). If feature 4 was black, however, the classifier favours class 3 and SOFS suggests to select a *different* feature next, namely feature 6 (c). After looking at it, the classifier now favors class 4 (d).

randomly chosen from a Bernoulli distribution with probability $p = 0.5$ to be either 0 or 1. Then, some normally distributed noise ϵ was added to the coordinates with $\epsilon \sim \mathcal{N}(0, 0.1)$. This places each of the points around one of eight corners of a cube, as shown in Figure 3. All other features carry no information and are initialised with a uniformly drawn value between 0 and 1. The goal is to classify each of the points into its correct cube corner. While it would be an easy task for any feature selection method to isolate the three information- carrying features from the random ones, we added one extra processing step to the dataset: The three coordinate features are not at the same position in each data point, but shifted depending on their class label. The index of the x-coordinate within a data point for class label c_i is $(i \mod |F|)$, y and z are positioned at $(i + 1 \mod |F|)$ and $(i + 2 \mod |F|)$, respectively. The modulo operator ensures that the coordinate indices are on valid positions, in case there are less then 10 features in the dataset.

This means that for class 1 with corner coordinate $(0, 0, 0)$, a data point would be $(x, y, z, r_1, r_2, r_3, \ldots)$, with r_i being the random features. For class 4 and corner coordinate $(0, 1, 1)$, the data points are defined as $(r_1, r_2, r_3, x, y, z, \ldots)$, and so on.

Conventional FS methods can only fail with this dataset (it is constructed that way). None of the features by itself is meaningful across all classes. The best they can achieve is to pick the 10 features that contain some coordinate information. While this seems to be a very unfair and artifical dataset, biased towards SOFS, one should keep in mind that it is quite common for features to only carry information

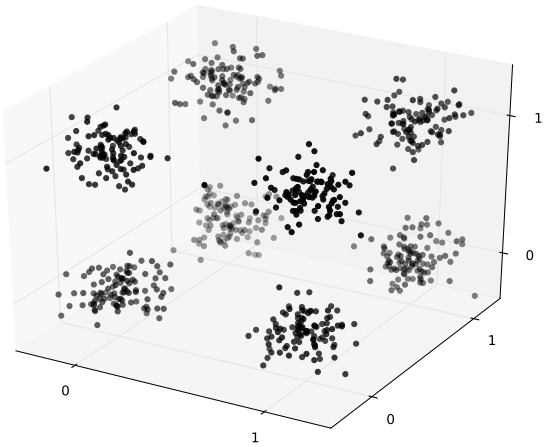

Fig. 3 Visualisation of the cube toy dataset. Each data point is assigned to one of the eight corners of a three-dimensional cube with a normally distributed noise in each dimension added. These three meaningful coordinate features are then combined with a number of non-informative random features. Furthermore, the indices of the coordinate features are different for each class.

in a certain class, or in other words, having class dependency. This experiment demonstrates that SOFS can use this class dependency to select those features during the classification process.

For classification, we use a logistic regression classifier with a softmax output for multiple classes. We tested datasets with 5, 10, 20 and 30 random features added in addition to the 3 meaningful coordinates. Each experiment was repeated 25 times with feature costs set to $r_k^- = 0.1\ \forall k$ and $r^+ = 1.0$ and we compare the development of the number of necessary features to access before correct classification occurs. Figure 5 shows the results over 350 episodes of training the SOFS agent. In all cases, the number of required feature quickly drops significantly below 10, the best possible outcome for conventional FS methods. Initial number of features and final number of features are displayed at the axis sides left and right respectively.

3.3 Handwritten MNIST digit classification

In this experiment we looked at the well-known MNIST handwritten digit classification task [**?**], consisting of $60,000$ training and $10,000$ validation examples. Each pattern is an image of 28×28 pixels of gray values in $[0, 1]$, the task is to map each image to one of the digits 0–9. We split every image into 16 non-overlapping 7×7 patches, each patch representing a feature.

We present results for a FFN as a non-sequential classifier and a RNN with Long Short Term Memory (LSTM) cells [**?**] as a sequential classifier with implicit memory. The FFN was chosen because it is a well-understood simple method, widely used for classification. The RNN was chosen to investigate, how naturally

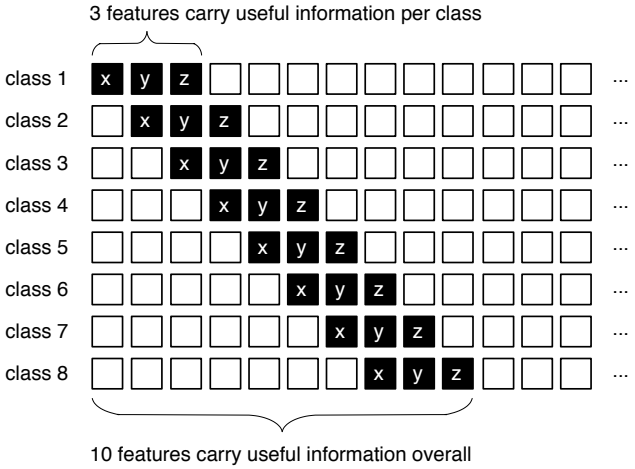

Fig. 4 Illustration of the coordinate feature placement for the cube dataset. The white boxes resemble random features which carry no useful information for the classification task. The black x, y, and z boxes are the informative features that describe a coordinate in three-dimensional space. The coordinate features are shifted by one position for each class. Conventional FS methods can only select features independent of the class and thus their best possible outcome is to choose the 10 features that carry useful information.

sequential classifiers work with SOFS. Throughout this experiment, rewards were set to $r^+ = 1.0$ and $r_k^- = -0.1\ \forall k$.

The FFN has one hidden layer with sigmoid activation, the architecture is 784-300-10. The output layer uses softmax activation with a 1-of-n coding. Pretraining of the classifier was executed online with a learning rate $\alpha = 0.1$ on the full training dataset. After 30 epochs of presenting all $60,000$ digits to the network, the error rate on the test dataset is 1.18%, slightly better than reported in [**?**]. However, this result is secondary, as the network acts merely as an environment for the SOFS agent. During SOFS training, each episode uses a random sample from the test dataset. Experience replay [**?**] is not used, as the LWPR function approximator is online in nature and can remember previous data. Figure 6 (left two plots) shows the development of episode lengths and returns during training of the SOFS agent. The average number of features required to correctly classify dropped from initially 7.65 (random order) to 3.06 (trained SOFS). The rate of incorrectly classified images was 0.77%.

The architecture of the RNN classifier is 49-50-10 with LSTM cells in the hidden layer. The output activation function is softmax with a 1-of-n coding. The RNN was pretrained with Back-Propagation Trough Time (BPTT), see e.g. [**?**], with a learning rate of $\alpha = 0.01$ and a random order of features. The results are illustrated in Figure 6 (right two plots). The average number of required features decreases from 4.91 features (random order) to 1.99 (trained SOFS). The rate of incorrectly classified images was 1.71%.

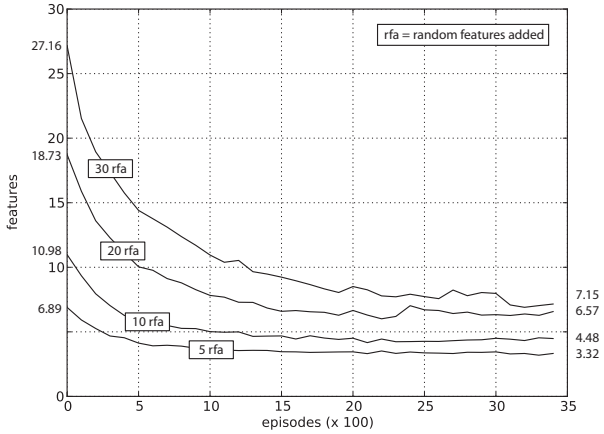

Fig. 5 Results of the SOFS training process on the cube dataset. The four plots show instances of the dataset with a different number of random features added (rfa) to the 3 informative features. In all cases, SOFS learns to ignore most of the random features and mostly focuses on the informative ones. However, the more random features were added, the more difficult it is to find the 3 coordinates. Initial number of features before training and final number of features are displayed at the axis sides left and right respectively.

3.4 Diabetes Dataset with Naive Bayes Classification

For the second experiment, we chose a more practical example from the medical field, the Pima Indians Diabetes data set [**?**]. We also decided on a Naive Bayes classification, to demonstrate the flexibility of the proposed method in terms of classifiers. The data set consists of 768 samples with 8 features (real-valued and integer) and two target classes (diabetes, no diabetes). Pretraining with a Naive Bayes classifier resulted in 73% correct prediction. There are two interesting aspects in this dataset. Firstly, it contains missing values, which should be handled well as we already use missing values to turn classification into a sequential process. Secondly, the features represent very different attributes of the (all female) patients. Some are simple questions (e.g. age, number of times pregnant), others are more complex medical tests (e.g. plasma glucose concentration after 2h in an oral glucose tolerance test). While the MNIST experiment used uniform costs r_k^-

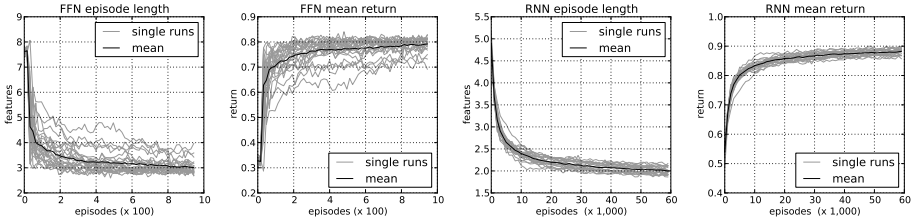

Fig. 6 Results of MNIST with FFN (left two plots) and RNN (right two plots). For each classifier, mean episode length and mean return over training episodes are shown.

Table 1 Assigned feature costs for diabetes dataset.

# pregnant	2h glucose concentr.	blood pressure	skin fold thickness	2h serum insulin	BMI	diabetes pedigree	age
-1	-120	-5	-5	-120	-5	-60	-1

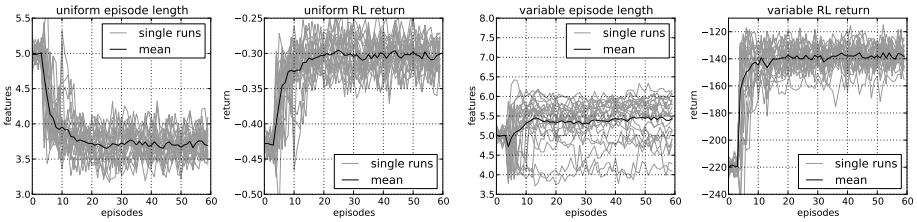

Fig. 7 Results of the PIMA diabetes dataset with Naive Bayes classification. Left two figures: episode lengths and mean returns for uniform feature costs. Right two figures: episode lengths and mean returns for feature costs according to Table 1.

for all features f_k, this experiment demonstrates another property of SOFS: the feature costs can be weighted, representing cheaper and more expensive features. To investigate the difference between uniform and variable feature costs, two sets of experiments were conducted: The first uses uniform costs $r_k^- = -0.1\,\forall k$, with a final number of required features of 3.7 on average. The second variant uses variable, estimated costs[3] shown in Table 1. Number of features *increased* from 4.99 to 5.66 on average, while the average return increased from -218 to -141. Figure 7 shows the results of both variants graphically.

3.5 Discussion

Early on we wanted to find out whether the SOFS agent is in fact able to learn to select features based on a current observation or if the selected features simply improve the results on average, independent of the belief. The results of the first toy experiment delivered an answer to that question: Figure 2 shows two cases (a) and (c) with different states, leading to the selection of feature 9 and feature 6, respectively. A first key finding is therefore, that SOFS is superior to any traditional FS method in that it can select features during a decision sequence *dependent* on the current observation.

The second toy dataset "Cube" was artificially created to demonstrate that features with class dependencies are a real issue for conventional FS methods. We didn't include any comparison to other FS methods but assumed that they would optimally choose the 10 features[4] that carried useful information overall. In all cases, SOFS could beat that number and reduce the amount of required features

[3] These costs represent a rough estimate of the time in minutes it takes to acquire the feature on a real patient. The estimates are based on oral communication with a local GP.

[4] with the exception of the 5rfa experiment, which only has 8 features in total. All of them carry information and an optimal static FS method would have to choose all 8.

to a significantly lower value by making the selection process dependent of the class belief.

The MNIST experiment with FFN classifier demonstrates a significant reduction of data consumption in two ways. Firstly, by making the decision process sequential, which enables the classifier to make decisions before all features have been looked at. This step alone reduces the average number of required features from all 16 features down to 7.65 (a reduction to 48%), and indicates that there is in fact a lot of redundancy in the MNIST images. Secondly, consumption is reduced further by learning the dependency of current belief and next feature, instead of accessing them in random order. After training the SOFS agent, data consumption decreases to 3.06 on average, 19% of the full data.

It is important to note that the stated error rates (1.18% for static and 0.77% for sequential classification) cannot be compared directly, because of the very different nature of the sequential approach. Sequential classification replaces the conventional error rates as performance measure based on the binary success of each sample (classified / not classified) with a scalar value (how many features until classified). In order to compare both classification methods, we would have to additionally learn when to stop the decision process, without using the class label. This could be achieved with a confidence threshold (e.g. if max(belief) reaches a certain value, as proposed in [**?**]) or by explicitly learning when to stop with either supervised or RL methods (the latter was successfully used in [**?**]). In this paper, we focussed on the RL feature selection process with existing classifiers rather than the performance of sequential classifiers. This issue will be addressed in a future publication.

Another aspect we investigated was the use of RNNs as naturally sequential classifiers. Where static classifiers still need to look at a full input (at least in terms of dimension, even though most of the pattern is filled with missing values), RNNs can make use of their intrinsic memory and achieve similar results with significantly fewer nodes in input and hidden layer and therefore even less data processing. They also converge with lower variance and reduce data consumption to a mere 12% on the MNIST task.

Finally, the Pima diabetes data set illustrates the use of variable feature costs, a variant that is naturally supported in our framework. The left two plots in Figure 7 show the development of episode length (i.e. number of selected features until correct classification) and mean return of the uniform cost experiment. As expected, episode lengths decrease with increasing returns, as the only objective for the agent is: *select those features first, that lead to correct classification.* However, if the reward scheme is changed (right two plots in Figure 7), we witness a *growth* of episode lengths in most of the 25 trials and on average. Still, all trials increase their returns (rightmost plot), which indicates that the agent does indeed learn and improve its performance. Comparing the final return average of -141 and the worst final return of -160 to the individual costs of Table 1, it becomes clear that in all runs, only one of the three most expensive features (number 2, 5 and 7) was selected. This behavior was caused by the different objective: *minimize the overall costs associated with the features.* In other words, it is okay to select many features, as long as they are cheap.

4 Conclusion

We have derived classification as a POMDP and thus made it accessible to RL methods. The application we focussed on was minimization of data consumption, by training an RL agent to pick features first that lead to quick classification. We presented results for different classifiers (both static and sequential) on vision and medical tasks. Our approach reduces the number of necessary features to access to a fraction of the full input, down to 12% with RNN classifiers. We also demonstrated that SOFS is able to deal with weighted feature costs, a property that exists in plenty of real-world applications. A new action selection method was introduced that draws actions without replacement. It should prove useful in other ordering tasks as well, such as scheduling problems. Lastly, we would like to point out that our approach is not limited to classification but easily extends to regression or other supervised tasks.