Sequential Feature Selection for Classification

Thomas Rückstieß[1], Christian Osendorfer[1], and Patrick van der Smagt[2]

[1] Technische Universität München, 85748 Garching, Germany,
{ruecksti,osendorf}@in.tum.de
[2] German Aerospace Center / DLR, 82230 Wessling, Germany,
smagt@dlr.de

Abstract. In most real-world information processing problems, data is not a free resource; its acquisition is rather time-consuming and/or expensive. We investigate how these two factors can be included in supervised classification tasks by deriving classification as a sequential decision process and making it accessible to Reinforcement Learning. Our method performs a sequential feature selection that learns which features are most informative at each timestep, choosing the next feature depending on the already selected features and the internal belief of the classifier. Experiments on 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

In recent times, an enormous increase in data has been observed, without a corresponding growth of the information contained within them. In other words, the *redundancy* of data continuously increases. An example of such effects can be found in medical imaging. Diagnostic methods can be improved by increasing the amount of MRI, CT, EMG, and other imaging data yet the amount of underlying information does not increase. Even worse, the redundancy of such data seems to negatively impact the performance of associated classification methods. Indeed, common engineering practices employ data-driven methods (including dimensionality reduction, nonlinear PCA, etc.) to reduce data redundancy.

On the other hand, obtaining qualitatively good data gets increasingly expensive. Again, medical data serves as a good example: not only do the costs of the above-mentioned medical imaging techniques explode—MRT scans are performed at the end user price of several thousands of US dollars per hour—but also diagnostics tests are getting increasingly intricate and therefore costly, to the point that a selection of the right diagnostic methods while maintaining the level of diagnostic certainty is of high value.

Also, from a computer scientist's perspective, the amount of processable data grows faster than processor speed. According to various studies[3], recent years

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

showed an annual 40–60% increase of commercial storage needs and a 40+-fold increase is expected in the next decade. Though this may, just like the integration density of processors, follow Moore's law, the increase of computer speed is well below that.

In short, an improved approach *feature selection* (FS) is needed, which not only optimally spans the input space, but optimizes with respect to data consumption. All of these arguments clearly demonstrate the advantage of carefully selecting relevant portions of data. Going beyond traditional FS methods, in this paper we lay out and demonstrate an approach of selecting features in sequence, making the decision which feature to select next *dependent* on previously selected features and the current internal state of the supervised method that it interacts with. In particular, our sequential feature selection (SFS) will embed Reinforcement Learning (RL) into classification tasks, with the objective to reduce data consumption and associated costs of features during classification. 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 high classification results?*"

Feature selection with RL has been addressed previously [5], 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 [17, 11] and attentional control policy learning [1, 14]. A similar concept, Online Streaming FS [17] 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 feature selection, our scenario differs in that it allows access to all features with the subgoal of minimizing data consumption. A similar approach to ours is outlined in [10], 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.

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, both on problems 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 one of a finite set of class labels $\mathcal{C}$. 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 want 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, which defeats the purpose of minimizing data access in most real-world scenarios. 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.

We believe that by turning classification into a sequential decision process, we can further significantly reduce the amount of data to process, as FS and classification then 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. This will be achieved by using a fully trained classifier as an environment for an RL agent, that learns which feature to access next, receiving reward on successful classification of the partially uncovered input pattern.

2.2 Sequential Classification

A first step towards our goal is to re-formulate classification as a Partially Observable Markov Decision Process[4] (POMDP), making the problem sequential and thus accessible to Reinforcement Learning algorithms. 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* $\text{powerseq}(M)$ of a set M to be the set of all permutations of all elements of the power set of M, including the empty sequence $()$. As an example, for $M = \{1, 2\}$, the resulting $\text{powerseq}(M) = \{(), (1), (2), (1{,}2), (2{,}1)\}$. During an episode, the feature history $h_t \in \text{powerseq}(F)$ 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.

Classifiers in general are denoted with the symbol K. We define a *sequential* classifier $\widetilde{K}$ to be a functional mapping from the power sequence of feature values to a set of classes, i.e., $\widetilde{K} : \text{powerseq}\left(\{f(x)\}_{f \in F}\right) \rightarrow \mathcal{C}$. An additional requirement is to process the sequence one input a time in an online fashion,

[4] 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.

rather than classifying the whole sequence at once, and to output a class label after each input. Therefore, $\widetilde{K}$ requires some sort of memory. Recurrent Neural Networks (RNN) [7] 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 sequence $(f_1(x), \ldots, f_t(x))$ up to time t is presented instead.

As it turns out, the above approach of providing explicit memory is not limited to sequential classifiers. Any classifier, that can handle *missing values* [13] can be converted to 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*. 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. At each timestep, 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 as well, that will remain unchanged and only act as an environment in which the SFS agent learns.

As we deal with a *partially observable* MDP, we need to extract an observation from the classifier, that summarizes the past into a stationary belief. 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.

To finally 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 2.3 describes, how this can be implemented practically. 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 $o_t = b(x, \widetilde{K}, h_{t-1}) = b(s_t)$. 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: $o_t = b(x, \widetilde{K}, h_{t-1}) = (p_1, p_2, \ldots, p_{|\mathcal{C}|})$.

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 next state $s_{t+1} = T_x(s_t, a_t) =$

$(x, \widetilde{K}, h_{t-1} \circ a_t)$ and next observation $o_{t+1} = b(s_{t+1})$. Lastly, the reward function $\mathcal{R}^a_{ss'}$ 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:

$$r_t = \begin{cases} r^+ + r^-_{a_t} & \text{if } \widetilde{K}\left((h_\tau(x))_{0<\tau\leq t}\right) = c \\ r^-_{a_t} & \text{else} \end{cases} \tag{1}$$

2.3 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:

$$p(a_t|o_t) = \frac{e^{Q(o_t,a_t)/\tau}}{\sum_a e^{Q(o_t,a)/\tau}}, \tag{2}$$

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.

2.4 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) [3]. 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 [12], Gaussian Processes [2], and others [9]. In this paper, we will use Locally Weighted Projection Regression (LWPR) [15] as the value function approximator of choice, as it is a fast robust online method that can handle large amounts of data.

Algorithm 1 Sequential Feature Selection (SFS)

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

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 SFS 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.

3 Experiments and Discussion

We evaluate the proposed method on two different datasets: the MNIST handwritten digits classification task, and a medical dataset for diabetes prediction. Each experiment was repeated 25 times, the plots below show single runs (gray) and the mean value over all runs (black).

3.1 Handwritten MNIST digit classification

In this experiment we looked at the well-known MNIST handwritten digit classification task [8], 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 an FFN as a non-sequential classifier and an RNN with Long Short Term Memory (LSTM) cells [6] as a sequential classifier with implicit memory. The FFN was chosen because it is a well-understood simple method,

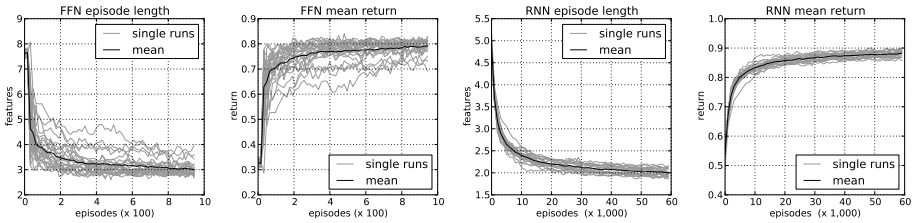

Fig. 1. Results of MNIST with FFN (left two plots) and RNN (right two plots). For each classifier, mean episode length and mean return over training episodes are shown.

widely used for classification. The RNN was chosen to investigate, how naturally sequential classifiers work with SFS. 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 [8]. However, this result is secondary, as the network acts merely as an environment for the SFS agent. During SFS training, each episode uses a random sample from the test dataset. Figure 1 (left two plots) shows the development of episode lengths and returns during training of the SFS agent. The average number of features required to correctly classify dropped from initially 7.65 (random order) to 3.06 (trained SFS). 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 Backpropagation Through Time (BPTT) (see, e.g., [16]), with a learning rate of $\alpha = 0.01$ and a random order of features. The results are illustrated in Figure 1 (right two plots). The average number of required features decreases from 4.91 features (random order) to 1.99 (trained SFS). The rate of incorrectly classified images was 1.71%.

3.2 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 [4]. 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)

Table 1. Assigned feature costs for diabetes dataset.

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

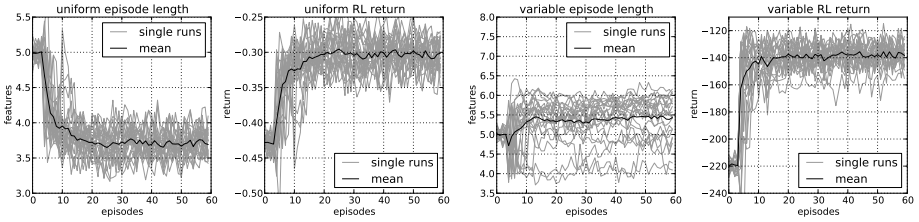

Fig. 2. Results of the PIMA diabetes dataset with Naive Bayes classification. Left two figures: episode lengths and mean returns for uniform feature costs. Right two figures: episode lengths and mean returns for feature costs according to Table 1.

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^- for all features f_k, this experiment demonstrates another property of SFS: 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[5] 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 2 shows the results of both variants graphically.

3.3 Discussion

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,

[5] 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.

instead of accessing them in random order. After training the SFS 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) or by explicitly learning when to stop with either supervised or RL methods. 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 2 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 2), 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 SFS 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.

References

1. L. Bazzani, N. de Freitas, H. Larochelle, V. Murino, and J.A. Ting. Learning attentional policies for tracking and recognition in video with deep networks. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.
2. M.P. Deisenroth, C.E. Rasmussen, and J. Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524, 2009.
3. D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(1):503, 2005.
4. A. Frank and A. Asuncion. UCI Machine Learning Repository. University of California, Irvine, CA. http://archive.ics.uci.edu/ml/, Oct. 2011.
5. R. Gaudel and M. Sebag. Feature selection as a one-player game. In *Proceedings of the 2nd NIPS Workshop on Optimization for Machine Learning*, 2009.
6. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
7. M. Hüsken and P. Stagge. Recurrent neural networks for time series classification. *Neurocomputing*, 50:223–235, 2003.
8. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
9. G. Neumann, M. Pfeiffer, and H. Hauser. Batch reinforcement learning methods for point to point movements. Technical report, Graz University of Technology, 2006.
10. E. Norouzi, M. Nili Ahmadabadi, and B. Nadjar Araabi. Attention control with reinforcement learning for face recognition under partial occlusion. *Machine Vision and Applications*, pages 1–12, 2010.
11. S. Perkins and J. Theiler. Online feature selection using grafting. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.
12. M. Riedmiller. Neural fitted Q iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. *Lecture notes in computer science*, 3720:317, 2005.
13. M. Saar-Tsechansky and F. Provost. Handling missing values when applying classification models. *Journal of machine learning research*, 8(1625-1657):9, 2007.
14. J. Schmidhuber and R. Huber. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems*, 2(1):135–141, 1991.
15. S. Vijayakumar and S. Schaal. Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.
16. R.J. Williams and J. Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501, 1990.
17. X. Wu, K. Yu, H. Wang, and W. Ding. Online streaming feature selection. In *Proceedings of the 27nd International Conference on Machine Learning*, 2010.