

Approximate Policy Iteration using Large-Margin Classifiers

Michail G. Lagoudakis and Ronald Parr

Duke University

Durham, NC 27708

{mgl, parr}@cs.duke.edu

Abstract

We present an approximate policy iteration algorithm that uses rollouts to estimate the value of each action under a given policy in a subset of states and a classifier to generalize and learn the improved policy over the entire state space. Using a multi-class support vector machine as the classifier, we obtained successful results on the inverted pendulum and the bicycle balancing and riding domains.

1 Introduction

Reinforcement learning provides an intuitively appealing framework for addressing a wide variety of planning and control problems. There has been significant success in tackling large-scale problems through the use of value function and/or policy approximation. The success of these methods, however, is contingent upon extensive and careful feature engineering, a common problem in most machine learning methods.

Modern classification methods have mitigated the feature engineering problems through the use of kernel methods, but very little has been done to exploit these recent advances for the purposes of reinforcement learning. Of course, we are not the first to note the potential benefits of modern classification methods to reinforcement learning. For example, Yoon *et al.* [2002] use inductive learning techniques, including boosting, to generalize across similar problems. Dietterich and Wang [2001] also use a kernel-based approximation method to generalize across similar problems. The novelty in our approach is the application of modern learning methods *within* a single, noisy problem at the inner loop of a policy iteration algorithm.¹ By using rollouts, we avoid the sometimes problematic step of value function approximation. Thus, our technique aims to address the critiques of value function methods raised by the proponents of direct policy search, while avoiding the confines of a parameterized policy space.

2 Definitions and Assumptions

A Markov decision process (MDP) is defined as a 6-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, D)$ where: $\mathcal{S}$ is the state space of the process;

¹Fern, Yoon, and Givan are also pursuing a similar approach.

$\mathcal{A}$ is a finite set of actions; P is a Markovian transition model, where $P(s, a, s')$ is the probability of making a transition to state s' when taking action a in state s ; R is a reward (or cost) function, such that $R(s, a)$ is the expected reward for taking action a in state s ; $\gamma \in [0, 1)$ is the discount factor for future rewards; and, D is the initial state distribution from which states are drawn when the process is initialized.

In reinforcement learning, it is assumed that the learner can observe the state of the process and the immediate reward at every step, however P and R are completely unknown. In this paper, we also make the assumption that our learning algorithm has access to a generative model of the process which is a black box that takes a state s and an action a as inputs and outputs a next state s' drawn from P and a reward r . Note that this is not the same as having the model (P and R) itself.

A policy π for an MDP is a mapping $\pi : \mathcal{S} \mapsto \mathcal{A}$ from states to actions, where $\pi(s)$ is the action the agent takes at state s . The value $V_\pi(s)$ of a state s under a policy π is the expected, total, discounted reward when the process begins in state s and all decisions at all steps are made according to policy π :

$$V_\pi(s) = E \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \mid s_0 = s \right] .$$

The goal of the decision maker is to compute or learn an optimal policy π^* that maximizes the expected total discounted reward from the initial state distribution:

$$\pi^* = \arg \max_{\pi} \eta(\pi, D) = E_{s \sim D} [V_{\pi^*}(s)] .$$

It is known that for every MDP, there exists at least one optimal deterministic policy.

3 (Approximate) Policy Iteration

Policy iteration (PI) is a method of discovering such a policy by iterating through a sequence of monotonically improving policies. Improvement at each iteration i is typically achieved by computing or learning the state-action value function Q_{π_i} of the current policy π_i , defined as

$$Q_{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s, a, s') V_{\pi_i}(s') ,$$

and then the improved policy as

$$\pi_{i+1}(s) = \arg \max_{a \in \mathcal{A}} Q_{\pi_i}(s, a) .$$

In practice, policy iteration terminates in a surprisingly small number of steps, however, it relies on exact representation of value functions and policies.

Approximate methods are frequently used when the state space of the underlying MDP is extremely large and exact (solution or learning) methods fail. A general framework of using approximation within policy iteration is known as *approximate policy iteration* (API). In its most general form, API assumes approximate representations of value functions ($\tilde{Q}$) and policies ($\hat{\pi}$). As a result, API cannot guarantee monotonic improvement and convergence to the optimal policy. A performance bound on the outcome of API can be constructed in terms of L_∞ bounds on the approximation errors [Bertsekas and Tsitsiklis, 1996]. In practice, API often finds very good policies in a few iterations, since it normally makes big steps in the space of possible policies. This is in contrast to policy gradient methods which, despite acceleration methods, are often forced to take very small steps.

4 Practical API without Value Functions

The dependence of typical API algorithms on approximate value functions places continuous function approximation methods at their inner loop. These methods typically minimize L_2 error, which is a poor match with the L_∞ bounds for API. This problem is *not* just theoretical. Efforts to improve performance, such as adding new features, can sometimes lead to surprisingly worse performance, making feature engineering a somewhat tedious and counterintuitive task.

An important observation, also noted by Fern, Yoon and Givan [2003], is that a Monte-Carlo technique, called *rollouts*, can be used within API to avoid the problematic value function approximation step entirely. Rollouts estimate $Q_{\pi_i}(s, a)$ from a generative model by executing action a in state s and following policy π_i thereafter, while recording the total discounted reward obtained during the entire trajectory. This simulation is repeated several times and the results are averaged over a large number of trajectories to obtain an accurate estimate $\tilde{Q}_{\pi_i}(s, a)$ of $Q_{\pi_i}(s, a)$. Rollouts were first used by Tesauro and Galperin [1997] for online improvement of a backgammon player. For our purposes, we can choose a representative set of states S_ρ (with distribution ρ over the state space) and perform enough rollouts to determine which action maximizes $\tilde{Q}_{\pi_i}(s, a)$ for the current policy. Rather than fitting a function approximator to the values obtained by the rollouts, we instead train a classifier on S_ρ , where each state is labelled with the maximizing action for the state. The algorithm is summarized in Figure 1.

LEARN is a supervised learning algorithm that trains a classifier given a set of labeled training data. The termination condition is left somewhat open ended. It can be when the performance of the current policy does not exceed that of the previous one, when two subsequent policies are similar (the notion of similarity will depend upon the learner used), or when a cycle of policies is detected (also learner dependent). If we assume a fortuitous choice of S_ρ and a sufficiently powerful learner that can correctly generalize from S_ρ to the entire state space, the algorithm at each iteration learns the improved policy over the previous one, effectively implement-

```

 $i \leftarrow 0$ 
 $\hat{\pi}_0 \leftarrow$  arbitrary policy (default: random)

repeat
  for each  $s \in S_\rho$ 
     $\tilde{Q}_{\hat{\pi}_i}(s, a) \leftarrow \text{ROLLOUT}(s, a, \hat{\pi}_i), \forall a \in \mathcal{A}$ 
     $\pi_{i+1}(s) = \arg \max_{a \in \mathcal{A}} \tilde{Q}_{\hat{\pi}_i}(s, a)$ 
   $\hat{\pi}_{i+1} \leftarrow \text{LEARN}\left(\left\{(s, \pi_{i+1}(s)) : s \in S_\rho\right\}\right)$ 
   $i = i + 1$ 
until termination

```

Figure 1: API with rollouts and classification.

ing a full policy iteration algorithm, and terminating with the optimal policy. However, for large-scale problems, choosing S_ρ and dealing with imperfect classifiers poses challenges.

We consider a number of alternative choices for the distribution ρ . A uniform distribution over the state space is a good choice for a low-dimensional state space, but it will result in poor coverage in high-dimensional spaces. A better choice would be a distribution that favors certain important parts of the state space over others. In classification, it is widely assumed that the classifier is trained on examples that are drawn from the same distribution it will be tested on. Therefore, representative states for learning policy π_{i+1} in iteration i should be drawn from the γ -discounted future state distribution [Jaakkola *et al.*, 1995] of the yet unknown policy π_{i+1} . Even though this policy is yet unknown, it is possible to closely approximate this distribution². Starting in some state s_0 drawn from D , a trajectory under π_{i+1} can be simulated by using rollouts repeatedly in each visited state to determine the actions of π_{i+1} . If the trajectory terminates with probability $(1 - \gamma)$, then states visited along such trajectories can be viewed as samples from the desired distribution.

The main contribution of this paper is a particular embodiment of the algorithm described in the previous section. For rollouts, we used basic statistical hypothesis testing (two-sample t -test) to determine statistically significant differences between the rollout estimates of $Q_{\hat{\pi}_i}$ for different actions. If action a is determined to be a clear winner for state s , then we treat (s, a) as a positive training example for state s , and all (s, a') , $a' \neq a$ as negative examples for s . Even if there is no clear winner action in some state s , negative examples can be still extracted for the clearly bad actions in s , allowing the classifier to choose freely any of the remaining (good) actions as the winner. The only case where no examples are generated is when all actions seem to be equally good.

The most significant contribution of effort is the use of support vector machines (SVMs) for LEARN. In our implementation we used the SVMTool package [Collobert and Bengio, 2001] as the multi-class classifier. With the kernel trick, SVMs are able to implicitly and automatically consider classifiers with very complex feature spaces.

²Special thanks to Alan Fern for sharing this observation.

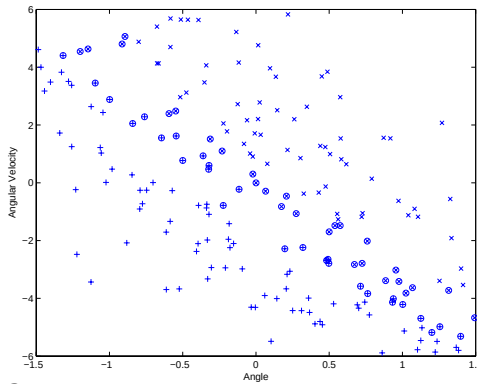


Figure 2: Positive(+), negative(x) examples; support vectors(o).

5 Experimental Results

In the *inverted pendulum* problem the goal is to balance a pendulum by applying forces to the left (LF), or to the right (RF), or no force (NF) at all. All actions are noisy. The state space is continuous and consists of the vertical angle and the angular velocity of the pendulum. Transitions are governed by the nonlinear dynamics of the system and the time step is 0.01 seconds. The reward is -1 when the angle exceeds $\pi/2$ in absolute value (end of the episode) and 0 otherwise. The discount factor is 0.95. Using about 200 states from ρ to perform rollouts, the algorithm consistently learns balancing policies in one or two iterations, starting from the random policy. The choice of ρ or the kernel (polynomial/Gaussian) did not affect the results significantly. Figure 2 shows the training data for the LF action with ρ being the uniform distribution. The number of support vectors was normally smaller than the number of rollout states. The constant C , the trade-off between training error and margin, was set to 1.

In the bicycle balancing and riding problem [Randløv and Alstrøm, 1998] the goal is to learn to balance and ride a bicycle to a target position located 1 km away from the starting location. The state description is a six-dimensional vector $(\theta, \dot{\theta}, \omega, \dot{\omega}, \psi, \dot{\psi})$, where θ is the angle of the handlebar, ω is the vertical angle of the bicycle, and ψ is the angle of the bicycle to the goal. The actions are the torque τ applied to the handlebar $\{-2, 0, +2\}$ and the displacement of the rider $v \{-0.02, 0, +0.02\}$. Actions are restricted so that either $\tau = 0$ or $v = 0$ giving a total of 5 actions. The noise is a uniformly distributed term in $[-0.02, +0.02]$ added to the displacement. The dynamics of the bicycle are based on the model of Randløv and Alstrøm [1998] and the time step is set to 0.02 seconds. As is typical with this problem, we used a shaping reward [Ng et al., 1999]. The reward r_t given at each time step was $r_t = (d_{t-1} - \gamma d_t)$, where d_t is the distance of the back wheel of the bicycle to the goal position at time t and γ is the discount factor which is set to 0.95.

Using a polynomial kernel of degree 5, we were able to solve the problem with uniform sampling of about 5000 rollout states. Sampling from the γ -discounted distribution of the target policy, the problem is solved with as few as 1000 rollout states, as shown in Figure 3 which shows ten sample trajectories of the bicycle in the two-dimensional plane from the initial position (0, 0) (left side) to the goal position

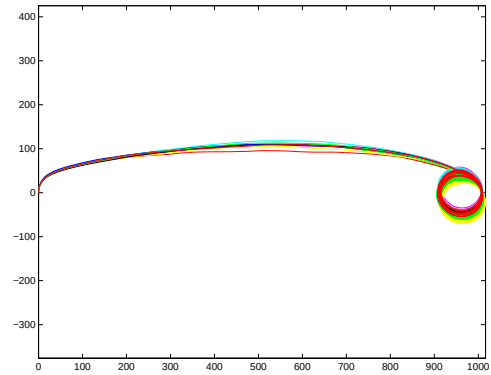


Figure 3: Successful trajectories of the bicycle.

(1000, 0) (right side). The input to the SVM was simply the six-dimensional state description and the value of C was 1. Performance was sensitive to the SVM parameters. We are currently doing further experimentation with larger numbers of rollout states and different kernel parameters.

This research was supported in part by NSF grant 0209088.

References

- [Bertsekas and Tsitsiklis, 1996] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996.
- [Collobert and Bengio, 2001] Ronan Collobert and Samy Bengio. SVMtorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research (JMLR)*, 1:143–160, 2001.
- [Dietterich and Wang, 2001] T. G. Dietterich and X. Wang. Batch value function approximation via support vectors. In *Advances in Neural Information Processing Systems 14*, 2001. MIT Press.
- [Fern et al., 2003] A. Fern, S. Yoon, and R. Givan. Approximately policy iteration with a policy language bias: Learning control policies in relational planning domains. In *Submitted to The Nineteenth International Conference on Machine Learning (ICML-2003)*, July 2003.
- [Jaakkola et al., 1995] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, Cambridge, Massachusetts, 1995. MIT Press.
- [Ng et al., 1999] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: theory and application to reward shaping. In *Proc. 16th International Conf. on Machine Learning*, pages 278–287. Morgan Kaufmann, San Francisco, CA, 1999.
- [Randløv and Alstrøm, 1998] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *The Fifteenth International Conference on Machine Learning*, 1998. Morgan Kaufmann.
- [Tesauro and Galperin, 1997] G. Tesauro and G. R. Galperin. On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing Systems (NIPS) 9*, 1997.
- [Yoon et al., 2002] S. W. Yoon, A. Fern, and B. Givan. Inductive policy selection for first-order MDPs. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, 2002. Morgan Kaufmann.

Active Learning with Ensembles for Image Classification

H. Liu and A. Mandvikar¹ and P. Foschi² and K. Torkkola³

¹Department of Computer Science & Engineering, PO Box 875406,
Arizona State University, Tempe, AZ 85287. {huanliu,amitm}@asu.edu

²Romberg Tiburon Center, 3152 Paradise Drive, Tiburon, CA 94920. tfoschi@sfsu.edu

³Human Interface Lab, Motorola Labs, Tempe, AZ 85284. kari.torkkola@motorola.com

Abstract

In many real-world tasks of image classification, limited amounts of labeled data are available to train automatic classifiers. Consequently, extensive human expert involvement is required for verification. A novel solution is presented that makes use of active learning combined with an ensemble of classifiers for *each class*. The result is a significant reduction in required expert involvement for uncertain image region classification.

1 Introduction

Multimedia contents are rapidly becoming a major target for data mining research. This work is concerned with image mining, discovering patterns and knowledge from images for the purpose of classifying images. The specific problem we address is image region classification. *Egeria Densa* is an exotic submerged aquatic weed causing navigation and reservoir-pumping problems in the Sacramento-San Joaquin Delta of Northern California. As a part of a control program to manage *Egeria*, classification of regions in aerial images is required. This problem can be abstracted to one of classifying massive data without class labels. Relying on human experts for class labeling is not only time-consuming and costly, but also unreliable if the experts are overburdened with numerous minute and routine tasks. Massive manual classification becomes impractical when images are complex with many different objects (e.g., water, land, *Egeria*) under varying picture-taking conditions (e.g., deep water, sun glint). The main objective of the project is to relieve experts from going through all the images and pointing out locations where *Egeria* exists in the image. We aim to automate the process via active learning to limit expert involvement to decisions about which the automatic classifier is uncertain.

The contributions of this work are a novel concept of *class-specific ensembles*, and learning class-specific ensembles. We notice that different types of classifiers are better suited to detecting different objects such as *Egeria*, land, water. Since it is impractical to train one classifier for each object (as experts need to provide training instances for all objects), we propose a novel approach to class-specific ensembles. We also discuss how to combine individual classifiers to learn class-specific ensembles. We show that this approach significantly reduces

the number of uncertain image regions and is better than a single ensemble for the task of object detection.

2 Class-Specific Ensembles

We may tend to use as many classifiers in an ensemble as possible because (1) each classification algorithm may have a different view of the training image and capture varied aspects of the image, as different classification algorithms have different biases and assumptions; and (2) no single classifier can cover all aspects. In other words, some algorithms may succeed in capturing some latent information about the domain, while others may fail. However, problems can result from using too many classification algorithms: e.g., (1) using more classification algorithms can result in longer overall training time, especially so if some of the algorithms are time-consuming to train; and (2) some algorithms may be prone to overfitting. If these algorithms are included in the ensemble, there may be a high risk of allowing the ensemble to overfit some training image(s). The above analysis suggests the necessity of searching for a relevant set of classifiers to form ensembles. Exhaustive search for the best combination is impractical because the search space is exponential in the total number of classification algorithms for consideration. Thus we need an efficient methodology to find the optimal combination of classifiers for the class-specific ensembles.

We adopt three performance criteria for this work: Precision, Recall, and number of uncertain regions (UC). The first two standard measures are defined in terms of the instances that are relevant and the instances that are classified (or retrieved). The third one is the number of instances (regions in an image) the class-specific ensembles cannot be certain about their class predictions. High precision or high recall alone is not a good performance measure as each describes only one aspect of classification. Together they provide a good measure: for example, the product of precision and recall (PR). If both values are 1, their product is 1, which means all and only positive instances are classified as positive. Hence, PR is a measure for both generality and accuracy. F measure can be used for the same purpose.

3 Learning Class-Specific Ensembles

In order to learn class-specific ensembles, we need a data set that links classifiers to the label of each image region. This

input: Tr, l_{Tr}, E : Set of n classification algorithms
output: $E_{l=yes}, E_{l=no}$

- 01 Train E with Tr to obtain n classifiers, cl_1 to cl_n ;
- 02 Obtain class labels, l_{Tr}^1 to l_{Tr}^n for Tr using cl_1 to cl_n ;
- 03 Form a data set, $D \leftarrow \{l_{Tr}^1, l_{Tr}^2, \dots, l_{Tr}^n, l_{Tr}\}$;
- 04 Learn association rules, $Assoc$ from D ;
- 05 $Assoc_1 \leftarrow \text{Filter}(Assoc / \text{consequent is } l_{Tr} = yes)$;
- 06 $Assoc_0 \leftarrow \text{Filter}(Assoc / \text{consequent is } l_{Tr} = no)$;
- 07 $m_{l=yes} \leftarrow \text{Max}(Assoc_1, \text{supp} * \text{conf})$;
- 08 $m_{l=no} \leftarrow \text{Max}(Assoc_0, \text{supp} * \text{conf})$;
- 09 $Assoc_1 \leftarrow \text{Filter}(Assoc_1 / \text{supp} * \text{conf} \geq 0.9 * m_{l=yes})$;
- 10 $Assoc_0 \leftarrow \text{Filter}(Assoc_0 / \text{supp} * \text{conf} \geq 0.9 * m_{l=no})$;
- 11 $Assoc_1 \leftarrow \text{Sort}(Assoc_1, \text{length}(\text{precedent}))$;
- 12 $Assoc_0 \leftarrow \text{Sort}(Assoc_0, \text{length}(\text{precedent}))$;
- 13 $E_{l=yes} \leftarrow \text{Precedent}(\text{First}(Assoc_1))$;
- 14 $E_{l=no} \leftarrow \text{Precedent}(\text{First}(Assoc_0))$;

Figure 1: Algorithm for selecting optimal classifiers

new data set can be obtained by applying all the classification algorithms to the training image so that each classifier is a feature (i.e., a column) and its value is the prediction of the classifier. For each image region (one instance in the new data set), there are predictions of all the classifiers and also the class label ‘Egeria’ or ‘non-Egeria’ given by experts. We are concerned only with those rules that have the class label Egeria or not as the consequent. We will restrict our search to such rules and obtain rules with the maximum number of features (classifiers) in the precedent without a significant loss in support or confidence. The best rule for each class label indicates the best combination of classifiers for the ensemble.

Among many learning algorithms for classification, clustering, and association rules, we observe that association rule algorithms [Agrawal and Srikant, 1994] can search the attribute space to find the best combination of attributes associated with a class. Our algorithm is presented in Figure 1. It takes as input the entire set of classification algorithms E and training data Tr with class labels l_{Tr} , and produces as output two ensembles for class label *yes* and class label *no*.

The next task is to use the dual ensembles ($E_{l=yes}$ and $E_{l=no}$) to determine certain and uncertain instances. We need to decide the maximum number of classifiers in an ensemble that should agree on a prediction to reach a decision of “certain” or “uncertain” for each ensemble. An ensemble with all classifiers being required to agree on a prediction would lead to high precision, but low recall; an ensemble with few classifiers being required to agree would lead to high recall and low precision. We need to find a balanced number of classifiers with which the ensemble gives the best estimated precision and recall. The training image is used again for this task.

The dual ensembles $E_{l=yes}$ and $E_{l=no}$ work together to decide if an instance’s prediction is certain or not following the rule of majority. In predicting an instance, if both $E_{l=yes}$ and $E_{l=no}$ are certain with their predictions, follow the one with more agreeing classifiers; if one is certain and the other

is uncertain, follow the certain one; if both are uncertain, the instance is uncertain.

4 Experiments and Evaluations

With the principal goal of reducing the burden on experts, we use only one image for training and apply the learned results to another 16 testing images of different areas for detection. Among the classification algorithms available in the machine-learning package WEKA [Witten and Frank, 2000], we select all that can be applied to the image domain to ensure a variety of classification algorithms. We apply the algorithm in Figure 1 with the complete set of classification algorithms as input. The class-specific ensembles found by association rules are: **Class = yes** $\Leftarrow$ Decision Trees C4.5, Alternating Decision Trees, Decision Trees (PART), PRISM, Hyper Pipes, Kernel Density, Logistic, Decision Tables; and **Class = no** $\Leftarrow$ Id3, Alternating Decision Trees, Decision Trees (PART), PRISM, Kernel Density, Instance Based1, Decision Tables.

We design 3 experiments and provide the summarized results here (the details will be provided upon request).

Experiment 1. We compare the use of dual ensembles ($E_{l=yes}$ and $E_{l=no}$) with that of single ensembles (either $E_{l=yes}$ or $E_{l=no}$). The average UCincrease (increase in uncertain regions) is almost 53% and the average PRgain is -1.04%. It is observed that in general, dual ensembles are not only more accurate, but also separate certain and uncertain instances better than single ensembles.

Experiment 2. We compare the learned dual ensembles to 10 randomly selected dual ensembles. Each classifier is randomly chosen and learns from the training image. Although the average PRgain is only decreased by 2.01%, the UCincrease increases significantly by 846.6%. Hence, it is necessary to search for relevant dual ensembles, as random dual ensembles work poorly in reducing UC.

Experiment 3. We compare the learned dual ensembles to results obtained by the classification rules of domain experts. The experts’ rules outperform the learned dual ensembles in terms of PRgain by 14.6%, but the number of uncertain instances (UC) increases by 408.4%. The high PRgain and high UC for the expert classification rules is due to the fact that an expert can only directly work on the former (designing highly general and accurate rules), but not on the latter (finding low UC rules). Our system is particularly designed to compensate in this shortcoming.

5 Conclusion

We present a novel approach to active learning with ensembles of classifiers. One ensemble is trained for *each class*. Extensive experiments were conducted in the real-world domain of detecting seaweed in aerial images.

References

- [Agrawal and Srikant, 1994] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. VLDB*, pages 487–499, 1994.
- [Witten and Frank, 2000] I. Witten and E. Frank. *Data Mining: Practical Machine Learning tools and techniques with java implementations*. Morgan Kauffmann, 2000.

Item Selection Strategies for Collaborative Filtering

Rachael Rafter, Barry Smyth

Smart Media Institute, University College Dublin, Ireland
{rachael.rafter, barry.smyth}@ucd.ie

1 Introduction

Automated collaborative filtering (ACF) methods leverage the ratings-based profiles of users that are similar to some target user in order to proactively select relevant items, or predictively rate specific items, for the target user. Many of the advantages of ACF methods are derived from its content-free approach to recommendation; it is not necessary to rely on content-based descriptions of the recommendable items, only their ratings distribution across the population of raters. Furthermore, ACF methods have an element of serendipity associated with them, as users can find items for which they would never have explicitly searched but nonetheless find interesting. The raters, items and ratings form a sparse matrix, R and the classical ACF prediction task is to predict the rating that user t will give to item i given that $R_{t,i}$ is currently empty. For example, Resnick's well-known algorithm [Resnick *et al.*, 1994] predicts $R_{t,i}$ based on t 's average rating, $\bar{t}$, and $R_{r,i}$, the rating each rater r gives to i , relative to r 's average rating, $\bar{r}$, and weighted by $C_{t,r}$, the correlation between the shared ratings of t and r .

$$R_{t,i} = \bar{t} + \frac{\sum_{r \neq t} ((R_{r,i} - \bar{r}) * C_{t,r})}{\sum_{r \neq t} |C_{t,r}|} \quad (1)$$

It is well known that the success of ACF depends broadly speaking, on the quality of the user profiles and on the coverage of the ratings space that they provide [Smyth *et al.*, 2002; Rafter and Smyth, 2001a; Rafter *et al.*, 2000a; 2000b; Herlocker *et al.*, 1999; Breese *et al.*, 1998; Konstan *et al.*, 1997]. At the same time efficiency depends on the number of users that are taken into account during rating and the size of their individual profiles. For example, Resnick's algorithm is $O(r \cdot i)$ where r is the number of raters and i the average number of items that they have rated.

In this paper we are interested in looking for ways to improve the efficiency of ACF without compromising the quality of recommendations. Our basic proposal is to find ways of reducing the number of items that need to be considered during recommendation. For example, Resnick's algorithm above compares each rater profile to the target user by computing the correlation between all item ratings that these profiles share, which can often be a large number of ratings. By selecting a subset of shared ratings we can significantly improve the efficiency of this fundamental prediction step.

However, naive item selection strategies are likely to result in significant reductions in recommendation accuracy and so more intelligent strategies, based on an understanding of the relative value of different items, are proposed.

2 Item Selection Strategies

Let us assume that for reasons of efficiency it is necessary to select a subset of k shared items for consideration during ACF prediction using Resnick's algorithm. Which subset should be chosen? One could select the k items at random. Alternatively one could look to identify those items that are likely to contribute more information to the prediction task.

2.1 Inverse Popularity

One approach to selecting useful items during ACF prediction is to look at how popular an item is among the rater population. The motivation here lies in the belief that relatively poorly distributed items among the population should be better predictors of a user's preferences [Rafter and Smyth, 2001b]; if a user has looked at an unusual item this should tell us more about her preferences than an item that everyone is familiar with. For example in a movie recommender, most people will have seen movies like "The Lord of the Rings" or "Fight Club" and therefore most profiles will contain these movies, but all this really tells us is that these are popular movies. However if a user profile contains more unusual movies such as "Run Lola Run" or "Buffalo 66", that most other profiles don't contain, then we can tell more about that user's *individual* taste for movies. Accordingly, our inverse popularity strategy weights the items in a profile according to the number of users in the population that have that item in their profile and selects those k items, shared between rater and target user, that have the highest inverse popularity score (Equation 2).

$$InvPop(i) = 1 - \frac{|r_j \in R : r_j(i) \neq \{\}|}{|R|} \quad (2)$$

2.2 Deviation of Ratings

The deviation of ratings strategy is based on the principle that the more users tend to disagree about an item the more informative that item is about any given user's particular preferences. For example, in a movie recommender most users will

give a movie like "The Shawshank Redemption" a high rating and so there is little to be gained by comparing users along this dimension. However, by the same token, there may tend to be a lot of disagreement about "Vanilla Sky", and knowing whether a user likes or dislikes this movie is likely to tell us a lot about this user.

One way to take account of the above idea is to weight items according to the standard deviation of their ratings across the user population. Items that have a high standard deviation in their ratings (r_i) (a lot of disagreement) get a high weight, and items with a lot of agreement (low standard deviation) get a low weight (Equation 3). Once again, using this strategy, we can prioritise the selection of the k shared items.

$$DevRating(i) = StdDev(i) = \sqrt{\frac{\sum_{r_i \in R} (r_i - \bar{r})^2}{|R| - 1}} \quad (3)$$

3 Experimental Evaluation

An experimental evaluation has been carried out to evaluate how our inverse popularity and deviation of ratings selection strategies impact prediction quality. To do this we have implemented a version of Resnick's ACF algorithm and modified it to select different sized subsets of shared items as the basis for prediction. Four different item selection strategies are reported:

- *All* selects all shared items as per the standard Resnick approach.
- *Random* selects a random subset of k shared items.
- *Popularity* selects a subset of k shared items according to the inverse popularity metric.
- *Deviation* selects a subset of k shared items according to the deviation of ratings metric.

In turn we evaluated the above methods using a subset of the largest 2050 profiles (2000 as the user population, and 50 as the target users) from the EachMovie dataset, [McJones, 1997]. From each target user profile we selected 10 items at random (500 in total) for which to predict scores measuring the quality of the prediction as the *average prediction error* by computing the difference between the predicted rating and the actual rating in the standard way. In addition, we vary the value of k from 5 to 100 to investigate the impact of different profile subset sizes.

The results are presented in Figure 1 as graphs of average prediction error versus k for each of the 4 selection strategies (All, Random, Popularity, Deviation). The results for All are obviously a straight line as this strategy is unaffected by k by design. The error for this strategy serves as an optimal benchmark against which to judge the other 3 techniques. In each case we find that the Popularity and Deviation techniques significantly outperform the naive Random selection approach. For example, at $k = 25$ the Random method presents with a prediction error of about 0.207 compared to errors of 0.185 and 0.186 for the Popularity and Deviation methods, and a minimum error of 0.183 for the All benchmark. A similar pattern is found for $k < 50$.

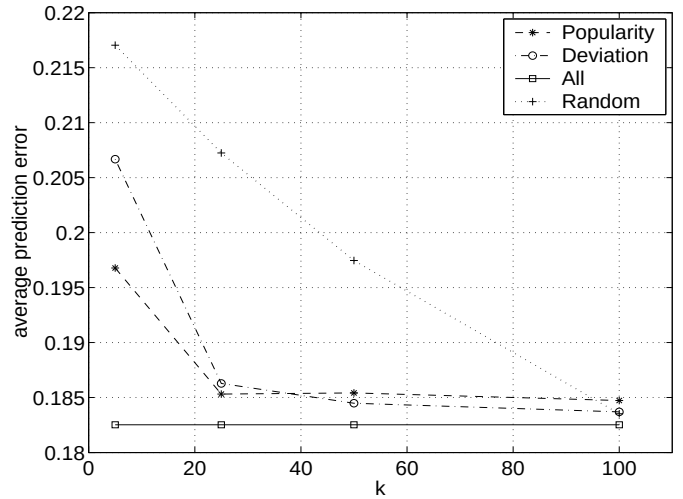


Figure 1: Average prediction error v.s. k

It is interesting to note that for $k = 100$ the prediction accuracy of the various techniques is seen to converge, indicating that there is little difference in the selection benefits for item subsets of this size. However, the average overlap of the population is 116.75 meaning that at the $k = 100$ mark almost all overlapping items are being considered by any of the four strategies and therefore it is appropriate that the accuracy converges. In point of fact it is appropriate that item selection strategies work well at low values of k for two reasons. First, it maximises the potential efficiency advantages. Secondly, most profiles tend to have relatively small numbers of ratings and so the shared ratings between two profiles is likely to be small; in EachMovie more than 90% of profiles have less than 100 ratings.

4 Conclusions and Future Work

We have described two item selection strategies for ACF (Deviation and Popularity) that reduce the set of items used when making predictions. We have discussed how these strategies improve the efficiency of ACF and our evaluation has shown that these improvements incur little compromise in the quality of prediction, when compared with the standard Resnick algorithm. The results also show that our strategies outperform the naive Random strategy and therefore are selecting a more intelligent subset of items. Efficiency in ACF is a well documented problem and we believe that item selection strategies such as those proposed in this paper are one of the keys to solving efficiency problems. In the future we look to investigating other similar item selection strategies (such as combining the Popularity and Deviation methods), based on a further understanding of the relative values of items.

The development of item selection strategies such as those we have described forms part of a larger research goal concerning the value of items in ACF. In any ACF prediction or recommendation task we can imagine three distinct beneficiaries. The first and most obvious beneficiary is the target user, the person for whom the prediction is being made. The second is the party making the recommendation - for example,

an online store whose desire it is to make recommendations to the target user and encourage them to purchase something. Finally the ACF system itself is also a potential beneficiary in the sense that with each prediction or recommendation made the system learns a new piece of information about the user. In such a situation where three distinct parties can potentially benefit from the outcome of the recommendation, there are clear trade-offs. For example, the online store may prefer to recommend expensive items over cheaper alternatives to customers and even weight items for recommendation not only according to their relevancy to the customer but also according to their price. Conversely, the user is unlikely to opt for such a recommendation strategy. The trade-off we are most interested in investigating is that of optimising recommendations for short-term benefit against optimising them for long-term benefit. The straightforward recommendation strategy is to select the item with the highest relevancy for the target user (i.e. the one she is most likely to like) for recommendation. This optimises recommendations on a short-term basis - the best option is selected for the current recommendation. However, if we were to weight recommendations not only based on the probability of the user liking them, but also based on the amount of knowledge the system is likely to gain with each recommendation we could potentially make better recommendations in the future.

Our research so far has shown that different items have different levels of knowledge gain. Relatively unpopular items, or those that tend to have a high deviation of ratings tell us more about the user and her preferences than an item that everyone has seen and liked. In the future we plan to look at the potential benefit of weighting items not only according to their relevance to the user, but also according to their associated knowledge gain, and investigating how such a strategy affects the quality of future predictions.

References

- [Breese *et al.*, 1998] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of 14th Conference on Uncertainty in Artificial Intelligence*, Madison, Wisconsin, USA, July 1998.
- [Herlocker *et al.*, 1999] J. Herlocker, J. Konstan, A. Borchers, and J. Reidl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Berkeley, California, USA, August 1999.
- [Konstan *et al.*, 1997] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Reidl. GroupLens: Applying collaborative filtering to usenet news. *Communications of ACM*, 40(3):77–87, March 1997.
- [McJones, 1997] P. McJones. Eachmovie collaborative filtering data set. Available from <http://research.compaq.com/SRC/eachmovie/>, 1997.
- [Rafter and Smyth, 2001a] R. Rafter and B. Smyth. Passive profiling from server logs in an online recruitment environment. In *Proceedings of IJCAI Workshop on Intelligent Techniques for Web Personalization (ITWP2001)*, Seattle, Washington, U.S.A., August 2001.
- [Rafter and Smyth, 2001b] R. Rafter and B. Smyth. Towards a domain analysis methodology for collaborative filtering. In *Proceedings of 23rd European Colloquium on Information Retrieval Research (ECIR01)*, Darmstadt, Germany, April 2001.
- [Rafter *et al.*, 2000a] R. Rafter, K. Bradley, and B. Smyth. Automated collaborative filtering applications for online recruitment services. In *Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-based Systems*, Trento, Italy, August 2000.
- [Rafter *et al.*, 2000b] R. Rafter, K. Bradley, and B. Smyth. Personalised retrieval for online recruitment services. In *Proceedings of the 22nd Annual Colloquium on Information Retrieval (IRSG2000)*, Cambridge, UK, April 2000.
- [Resnick *et al.*, 1994] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Reidl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM Conference on Computer-Supported Cooperative Work (CSCW94)*, Chapel Hill, North Carolina, USA, August 1994.
- [Smyth *et al.*, 2002] B. Smyth, K. Bradley, and R. Rafter. Personalization techniques for online recruitment services. *Communications of the ACM*, 45(5):39–40, 2002.

Modular self-organization for a long-living autonomous agent

Bruno SCHERRER

scherrer@loria.fr

LORIA, BP 239

54506 Vandœuvre-lès-Nancy

France

Abstract

The aim of this paper is to provide a sound framework for addressing a difficult problem: the automatic construction of an autonomous agent's modular architecture. We briefly present two apparently uncorrelated frameworks: Autonomous planning through Markov Decision Processes and Kernel Clustering. Our fundamental idea is that the former addresses autonomy whereas the latter allows to tackle self-organizing issues. Relying on both frameworks, we show that modular self-organization can be formalized as a clustering problem in the space of MDPs. We derive a modular self-organizing algorithm in which an autonomous agent learns to efficiently spread n planning problems over m initially blank modules with $m < n$.

Introduction

This paper addresses the problem of building a long-living autonomous agent; by *long-living*, we mean that this agent has a large number of relatively complex and varying tasks to perform. Biology suggests some ideas about the way animals deal with a variety of tasks: brains are made of specialized and complementary areas/modules; skills are spread over modules. On the one hand, distributing functions and representations has immediate advantages: parallel processing implies reaction speed-up; a relative independence between modules gives more robustness. Both properties might clearly increase the agent's efficiency. On the other hand, the fact of distributing a system raises a fundamental issue: how does the organization process of the modules happen during the life-time?

There has been much research about the design of modular intelligent architectures (e.g. [Theoharous *et al.*, 2000] [Hauskrecht *et al.*, 1998] [Kaelbling, 1993]). It is nevertheless very often the (human) designer who decides the way modules are connected to each other and how they behave with respect to the others. Few works study the construction of these modules. To our knowledge, there are no effective works about modular self-organisation except for reactive tasks (stimulus-response associations) (e.g. [Jacobs *et al.*, 1991] [Digney, 1996]).

This paper proposes an algorithm by which the organization of an agent in functional modules is automatically computed. The most significant aspect of our work is that the number m of modules is fewer than the number n of tasks to be performed. Therefore, the approach we propose involves a high-level clustering process, in which the n tasks need to be "properly" spread over the m modules.

Section 1 introduces what we consider as the theoretical foundation for modelling a mono-task autonomous agent: Markov Decision Processes. Section 2 presents the Kernel Clustering approach: we consider this approach as a theoretical basis for addressing self-organization. Finally, Section 3 combines both domains in order to propose a modular self-organizing algorithm.

1 Modelling A Mono-Task Autonomous Agent

Markov Decision processes [Puterman, 1994] provide the theoretical foundations of challenging problems such as planning under uncertainty and reinforcement learning [Sutton and Barto, 1998]. They stand for a fundamental model for sequential decision making and they have been applied to many real world problems [Sutton, 1997]. This section describes this formalism and presents a general scheme for approaching difficult problems (that is problems in large domains).

A Markov Decision Process (MDP) is a controlled stochastic process satisfying the Markov property with rewards (numerical values) assigned to state-control pairs. Formally, an MDP is a four-tuple $\langle S, A, T, R \rangle$ where S is the *state space*, A is the *action space*, T is the *transition function* and R is the *reward function*. T is the state-transition probability distribution conditioned by the control:

$$T(s, a, s') \stackrel{\text{def}}{=} \Pr(s_{t+1} = s' | s_t = s, a_t = a) \quad (1)$$

$R(s, a) \in \mathbb{R}$ is the instantaneous reward for taking action $a \in A$ in state s .

The usual *MDP problem* consists in finding an *optimal policy*, that is a mapping $\pi : S \rightarrow A$ from states to actions, that maximises the following performance criterion, also called value function of policy π :

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \cdot R(s_t, \pi(s_t)) | s_0 = s \right] \quad (2)$$

It is shown that there exists a unique optimal value function V^* ; once V^* is computed, an optimal policy can immediately be derived (e.g. see [Puterman, 1994]).

In brief, solving an MDP problem amounts to computing the optimal value function. Well-known algorithms for doing so are *Value Iteration* and *Policy Iteration* (see [Puterman, 1994]). Their temporal complexity dramatically grows with the number of states [Littman *et al.*, 1995], so they can only be applied in small domains.

In large domains, it is impossible to solve an MDP exactly, so one usually addresses a complexity/quality compromise through an approximation scheme. Ideally, an approximation scheme for MDPs should consist of a set of *tractable* algorithms for

- computing an approximate optimal value function
- evaluating (an upper bound of) the approximation error
- improving the quality of approximation (by reducing the approximation error) while constraining the complexity.

The first two points are the fundamental theoretical bases for sound approximation. The third one is often interpreted as a learning process and corresponds to what most Machine Learning researchers study. For convenience, we respectively call these three procedures *Approximate()*, *Error()* and *Learn()*. Then, the practical use of an approximate scheme can be sketched by algorithm 1. One successively applies the

Algorithm 1 A general approximation scheme for a large state MDP

Input: a large state space MDP $\mathcal{M}$ and an initial approximation $\widehat{\mathcal{M}}$.

Output: a good approximate value function $\widehat{V}^*$.

```

while Error( $\widehat{\mathcal{M}}$ ,  $\mathcal{M}$ ) goes on diminishing do
     $\widehat{\mathcal{M}} \leftarrow \text{Learn}(\widehat{\mathcal{M}}, \mathcal{M})$ 
end while
 $\widehat{V}^* \leftarrow \text{Approximate}(\widehat{\mathcal{M}})$ 

```

Learn() procedure in order to minimize the approximation error; when this is done, one can compute a good approximate optimal value function.

2 Kernel Clustering

Before addressing the problem of modular self-organization, we need to present the Kernel Clustering paradigm. In [Diday, 1973], the author introduces the Kernel Clustering approach as an abstract generalization of vector quantization. Indeed, the author argues that, in general, a clustering problem is based on three elements:

- $(x_i)_{i \in I}$: a set of data points taken from a data space X
- $\{L_1, \dots, L_m\}$: a set of kernels taken from a kernel space $\mathcal{L}$
- $d : X \times \mathcal{L} \rightarrow \mathbb{R}^+$: A distance measure between any data point and any kernel. The smaller the distance $d(x, L)$, the more L is *representative* of the point x .

Given a set of kernels $\{L_1, \dots, L_m\}$, a data point x is naturally associated to its most *representative* kernel $L(x)$, i.e. the one that is the closest according to distance d :

$$L(x) = \operatorname{argmin}_{L \in \{L_1, \dots, L_m\}} d(x, L) \quad (3)$$

Conversely, a set of kernels $\{L_1, \dots, L_m\}$ naturally induces a partition of the data set $(x_i)_{i \in I}$ into m classes $\{C_1, \dots, C_m\}$, each class corresponding to a kernel:

$$\forall j \in (1, \dots, m), C_j = \{(x_i)_{i \in I}; L(x_i) = L_j\} \quad (4)$$

Given a data space, a data set, a kernel space and a distance $d()$, the goal of the Kernel Clustering problem is to find the set of kernels $\{L_1^*, \dots, L_m^*\}$ that minimizes the distortion D for the data set $(x_i)_{i \in I}$, which is defined as follows:

$$D = \sum_{i \in I} d(x_i, L(x_i)) = \sum_{j=1}^m \sum_{x \in C_j} d(x, L_j) \quad (5)$$

A general procedure for suboptimally solving this problem is known as the Dynamic Cluster algorithm [Diday, 1973], which we present in an online version in algorithm 2. It is

Algorithm 2 The on-line dynamic cluster algorithm

Input: A data set $(x_i)_{i \in I}$

Output: A set of kernels $\{L_1, \dots, L_m\}$ that optimizes the clustering (i.e. that minimizes the distortion)

Initialization:

Let $\{L_1, \dots, L_m\}$ be any set of kernels

Iterations:

while the distortion goes on diminishing **do**

Randomly pick a data point x from the data set

Find the kernel the most representative of x :

$$L \leftarrow L(x) = \operatorname{argmin}_{L' \in \{L_1, \dots, L_m\}} d(x, L')$$

Update L so that $d(x, L)$ diminishes

end while

a very intuitive process: for each piece of data x , one finds its most representative kernel L , and one updates L so that it gets even more representative of x . Little by little, one might expect that such a procedure will minimize the global distortion and eventually give a good clustering.

3 Modular Self-Organization

This final section shows how the Kernel Clustering paradigm can be used to formalize a modular self-organization problem in the MDP framework, the algorithmic solution of which will be given by the on-line Dynamic Cluster procedure (algorithm 2).

If one carefully compares the general learning scheme we have described in order to address a large state space MDP (algorithm 1) and the on-line Dynamic Cluster procedure (algorithm 2), one can see that the former is a specific case of the latter. More precisely, algorithm 1 solves a simple Kernel Clustering problem where

- the data space is the space of all possible MDPs and the data set is a unique task corresponding to an MDP $\mathcal{M}$

- the kernel space is the space of all possible approximations and there is one and only one kernel: $\widehat{\mathcal{M}}$
- the distance d is the $Error()$ function.

It is then straightforward to extend this simple clustering problem to a more general one (with n tasks/data points and m approximate models/kernels). Given a set of m approximate models $\{\widehat{\mathcal{M}}_1, \dots, \widehat{\mathcal{M}}_m\}$, an MDP $\mathcal{M}$ is naturally associated to the approximate model $\widehat{\mathcal{M}}(\mathcal{M})$ that makes the small error:

$$\widehat{\mathcal{M}}(\mathcal{M}) = \operatorname{argmin}_{\widehat{\mathcal{M}}' \in \{\widehat{\mathcal{M}}_1, \dots, \widehat{\mathcal{M}}_m\}} Error(\widehat{\mathcal{M}}', \mathcal{M}) \quad (6)$$

As before, a set of approximate models $\{\widehat{\mathcal{M}}_1, \dots, \widehat{\mathcal{M}}_m\}$ naturally induces a partition of any set of n MDPs $(\mathcal{M}_i)_{1 \leq i \leq n}$ into m classes $\{C_1, \dots, C_m\}$, each class corresponding to an approximate model:

$$\forall j \in (1, \dots, m), C_j = \{(\mathcal{M}_i)_{1 \leq i \leq n}; \widehat{\mathcal{M}}(\mathcal{M}_i) = \widehat{\mathcal{M}}_j\} \quad (7)$$

The transposition of the on-line Dynamic Cluster Algorithm into the MDP framework (algorithm 3) therefore allows to find a set of m approximate models that globally minimize the approximation error for n MDPs:

$$\sum_{i=1}^n Error(\widehat{\mathcal{M}}(\mathcal{M}_i), \mathcal{M}_i) = \sum_{j=1}^m \sum_{\mathcal{M} \in C_j} Error(\widehat{\mathcal{M}}(\mathcal{M}), \mathcal{M}) \quad (8)$$

In other words, it optimizes the tuning of m modules in

Algorithm 3 Modular Self-Organization

Input: A set of MDPs $(\mathcal{M}_i)_{i \in I}$

Output: A set of approximate models $(\widehat{\mathcal{M}}_1, \dots, \widehat{\mathcal{M}}_m)$ that globally minimizes the approximation error

Initialization:

Let $(\mathcal{M}_1, \dots, \mathcal{M}_m)$ be any set of approximate models

Iterations:

while the global approximation error goes on diminishing **do**

Randomly pick a task $\mathcal{M}$ from the set of MDPs

Find the best module for solving $\mathcal{M}$:

$$\widehat{\mathcal{M}} \leftarrow \operatorname{argmin}_{\widehat{\mathcal{M}}' \in \{\widehat{\mathcal{M}}_1, \dots, \widehat{\mathcal{M}}_m\}} Error(\widehat{\mathcal{M}}', \mathcal{M})$$

$$\widehat{\mathcal{M}} \leftarrow Learn(\widehat{\mathcal{M}}, \mathcal{M})$$

end while

order to efficiently solve n tasks, or, as we might say, it self-organizes the m modules in order to improve the resolution of the n tasks.

Conclusion

In this paper, we have described a general scheme for addressing large state space Markov Decision Processes. We have then showed how such an approach could be extended to an interesting problem: modular self-organization. Indeed, we have formalized modular self-organization as a clustering

problem in the space of MDPs. A natural algorithmic solution to this clustering problem (algorithm 3) uses an on-line version of the Dynamic Cluster algorithm (algorithm 2). Due to lack of space, we could not show any experimental evaluation; interested readers will find some in [Scherrer, 2003b] and [Scherrer, 2003a].

References

- [Diday, 1973] E. Diday. The dynamic clusters method and optimization in non hierarchical-clustering. In SpringerVerlag, editor, *5th Conference on optimization technique, Lecture Notes in Computer Science 3*, pages 241–258, 1973.
- [Digney, 1996] B. Digney. Emergent hierarchical control structures: Learning reactive hierarchical relationships in reinforcement environments, 1996.
- [Hauskrecht et al., 1998] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of Markov Decision Processes using macro-actions. In *Uncertainty in Artificial Intelligence*, pages 220–229, 1998.
- [Jacobs et al., 1991] R. Jacobs, M. Jordan, and A. Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science*, 15:219–250, 1991.
- [Kaelbling, 1993] L. P. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *International Conference on Machine Learning*, pages 167–173, 1993.
- [Littman et al., 1995] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 394–402, Montreal, Québec, Canada, 1995.
- [Puterman, 1994] M. Puterman. *Markov Decision Processes*. Wiley, New York, 1994.
- [Scherrer, 2003a] B. Scherrer. *Apprentissage de représentation et auto-organisation modulaire pour un agent autonome*. PhD thesis, Université Henri Poincaré - Nancy 1, January 2003.
- [Scherrer, 2003b] Bruno Scherrer. Modular Self-Organization for a long-living autonomous agent. Technical report, INRIA, April 2003.
- [Sutton and Barto, 1998] R.S. Sutton and A.G. Barto. *Reinforcement Learning, An introduction*. Bradford Book. The MIT Press, 1998.
- [Sutton, 1997] Richard S. Sutton. On the significance of markov decision processes. In *ICANN*, pages 273–282, 1997.
- [Theocharous et al., 2000] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning and planning with hierarchical stochastic models for robot navigation. In *ICML 2000 Workshop on Machine Learning of Spatial Knowledge*, Stanford University, July 2000.

Towards a Theoretical Framework for Ensemble Classification

Alexander K. Seewald

Austrian Research Institute for Artificial Intelligence

Freyung 6/VI/7, A-1010 Wien, Austria

alexsee@oefai.at; alex@seewald.at

Abstract

Ensemble learning schemes such as AdaBoost and Bagging enhance the performance of a single classifier by combining predictions from multiple classifiers of the same type. The predictions from an ensemble of diverse classifiers can be combined in related ways, e.g. by voting or simply by selecting the best classifier via cross-validation – a technique widely used in machine learning. However, since no ensemble scheme is always the best choice, a deeper insight into the structure of meaningful approaches to combine predictions is needed to achieve further progress. In this paper we offer an operational reformulation of common ensemble learning schemes – Voting, *Selection by Crossvalidation* (X-Val), Grading and Bagging – as a Stacking scheme with appropriate parameter settings. Thus, from a theoretical point of view all these schemes can be reduced to Stacking with an appropriate combination method. This result is an important step towards a general theoretical framework for the field of ensemble learning.

1 Introduction

We apologize that space restrictions are forcing us to be terse. For a more detailed version, see [Seewald, 2003]. We will first explain how Stacking works in order to lay the foundations for our functional definitions of meta classifiers later on. Fig. 1 shows Stacking on a hypothetical dataset.

During training, all base classifiers are evaluated via cross-validation on the original dataset. Each classifier’s output is a class probability distribution for every example, see Fig. 1(b). The concatenated class probability distributions of all base classifiers, followed by the class value, forms the meta-level training set for Stacking’s meta classifier, see Fig. 1(c). After training the meta classifier, the base classifiers are retrained on the complete training data.

For testing, the base classifiers are queried for their class probability distributions. These form a meta-example for the meta classifier which outputs the final class prediction.

2 Definitions

An arbitrary training dataset with n examples and k classes, and a single test instance, is given. N arbitrary base classifiers have been cross-validated on this dataset, and afterwards retrained on the whole dataset. All base classifiers output class probability distributions, i.e. estimated probabilities for each class instead of deciding on a single class.

Then, $P_{i,jk}$ refers to the probability given by classifier i for class j on example number k during CV. $\bar{P}_{ik}$ is the class probability distribution of classifier i on instance k . $\bar{P}_i$ refers to

AttrVec		Cl.
AttrVec ₁	a	
AttrVec ₂	b	
AttrVec ₃	c	
⋮	⋮	⋮
AttrVec _n	a	

(a) original training set

a	b	c
0.90	0.05	0.05
0.15	0.70	0.15
0.20	0.20	0.60
⋮	⋮	⋮
0.80	0.10	0.10

(b) sample class prob.dist.

Classifier ₁			⋮	Classifier _N			Cl.
a	b	c		a	b	c	
P _{1,a1}	P _{1,b1}	P _{1,c1}	⋮	P _{N,a1}	P _{N,b1}	P _{N,c1}	a
P _{1,a2}	P _{1,b2}	P _{1,c2}	⋮	P _{N,a2}	P _{N,b2}	P _{N,c2}	b
P _{1,a3}	P _{1,b3}	P _{1,c3}	⋮	P _{N,a3}	P _{N,b3}	P _{N,c3}	c
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
P _{1,an}	P _{1,bn}	P _{1,cn}	⋮	P _{N,an}	P _{N,bn}	P _{N,cn}	a

(c) training set for Stacking’s meta classifier

Figure 1: Illustration of Stacking on a dataset with three classes (a , b and c), n examples and N base classifiers. $P_{i,jk}$ refers to the probability given by classifier i for class j on example number k .

the class prob.dist. for classifier i on the unknown test instance. A fixed arbitrary order on the class values and N base classifiers is assumed. $Class_k$ is the true class for instance k . $AttrVec_k$ is the attribute vector of instance k . n is the number of instances. All indices are zero-based, e.g. the instance id k satisfies the equation $0 \leq k \leq n - 1$.

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases} \quad (1)$$

$$\arg \max_i (A_i) \equiv \min \{i \mid A_i = \max_{\forall i} (A_i)\} \quad (2)$$

As we mentioned, we assume that all ensemble learning schemes return class probability distributions. If predictions are needed, the position of the maximum class probability in the vector – i.e. the predicted class – is easily obtained via formula (2). Trivially, Stacking with predictions can also be simulated by this variant; simply by transforming the class distributions meta-dataset to predictions via (2) prior to applying the meta classifier.

We can now characterize every ensemble learning scheme by what features it extracts from the meta-dataset during training and how these features define the mapping from meta-dataset to final class probability distribution.

2.1 Voting

Voting, a straight-forward extension of voting for distribution classifiers, is the simplest case. During training, no features are extracted from the meta-dataset. In fact Voting does not even need the internal crossvalidation. During testing, Voting determines the final class probability distribution as follows.

$$\overline{pred} = \sum_{i=1}^N \frac{\bar{P}_i}{N} \quad (3)$$

Thus, it can be easily seen that the meta-classifier defined by just computing the mean probability distribution of the base classifiers – as above – simulates Voting with probability distributions. Voting with predictions can be mapped similarly. In this case, we use $\overline{P}_i$ instead of $\overline{P}_i$ in formula (3) $\overline{P}_i$ is the vector of P'_{ij} for all j , where

$$P'_{ij} = \begin{cases} 1 & \text{if } j = \arg \max_j (P_{ij}), \text{ for given } i \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

2.2 X-Val

For X-Val, which chooses the best base classifier on each fold by an internal ten-fold CV, we first determine the accuracy per classifier, which can be computed directly from the meta-level dataset, see (6) Then, we derive as feature the id of the classifier with the maximum accuracy. Thus, the value of *bestC* corresponds to our learned model.

$$\text{bestC} = \arg \max_i (Acc_i) \quad (5)$$

$$Acc_i = \frac{1}{n} \sum_{k=1}^n \delta(\arg \max_j (P_{ijk}), Class_k) \quad (6)$$

During testing, X-Val simply returns *bestC*'s prediction.

$$\overline{pred} = \overline{P}_{\text{bestC}} \quad (7)$$

2.3 Grading

For Grading [Seewald and Fürnkranz, 2001], the case is difficult. While we cannot simulate the original Grading, we will simulate a competitive variant equivalent to accuracy-weighted voting. More details see [Seewald, 2003].

During training, the accuracies of base classifiers are extracted using formula (6) The accuracies of all our base classifiers correspond to our learned model. During testing, we build the combined class probability distribution similar to Voting using predictions but with an additional weight – namely the accuracy we extracted earlier.

$$\overline{pred} = \frac{1}{\sum_{i=1}^N Acc_i} \sum_{i=1}^N \left[Acc_i \frac{\overline{P}_i}{N} \right] \quad (8)$$

where $\overline{P}_i$ is again the vector of P'_{ij} for all j .

2.4 Bagging

Bagging [Breiman, 1996] is another common ensemble technique. Here, the same type of classifier is repeatedly trained on new datasets, which have been generated from the original dataset via random sampling with replacement. Afterwards, the component classifiers are combined via simple unweighted voting.

Thus, we use same meta-classifier as for Voting with predictions. The number of base classifiers is equal to the iteration parameter of bagging – each base classifier for Stacking corresponds to one instantiation of the base learner for bagging. In order to simulate the random sampling from the training set, the base learner's training sets have to be modified before training, via formula (9)

$$nTS = \{[AttrVec_{ij}, Class_{ij}] | i_j = rand(0, n-1), \text{ for } \forall 0 \leq j \leq n-1\} \quad (9)$$

During training, Formula (9) is used to create – for each base classifier separately – a training set of the same size as the original training set via random sampling from the original training set, exactly as in Bagging. These training sets are then used to train the base classifiers. This approach can also be seen as a probabilistic wrapper around each base classifier. No features are extracted from the meta-level dataset during training, as for Voting.

During testing, each base classifier gives a prediction. These predictions are then voted to yield the final prediction, exactly as for Voting with predictions, i.e. (3) modified via (4) – for more details refer to subsection 2.1. Concluding, we have shown the equivalence of Bagging and Stacking.

3 Discussion & Conclusion

By definition StackingC [Seewald, 2002], can also be mapped onto Stacking via a special meta classifier. In fact, the available implementation is a specialized subclass of a common meta classifier, MLR. Another recent variant, SMM5 [Dzeroski and Zenko, 2002], is also implemented via a special meta classifier and thus amenable to the same kind of mapping. However, AdaBoost [Freund and Schapire, 1996] cannot be simulated by Stacking because of its sequential nature.

While the given formal definitions of meta classifiers are mainly intended to enable further theoretical work, a direct implementation is also feasible. The cost penalty for the simulation is small, since training the meta classifiers usually contributes little to the total runtime.

Concluding, we have shown that Stacking is equivalent to common ensemble learning schemes, namely *Selection by Crossvalidation* (X-Val), Voting of either class probability distributions or predictions, a competitive variant of Grading, and Bagging. Recent variants such as StackingC [Seewald, 2002] and SMM5 [Dzeroski and Zenko, 2002] are also equivalent. Thus we conclude that our approach offers a unique viewpoint on Stacking which is an important step towards a theoretical framework for ensemble learning.

Acknowledgements This research is supported by the Austrian Fonds zur Förderung der Wissenschaftlichen Forschung (FWF) under grant no. P12645-INF. The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry of Education, Science and Culture.

References

- [Breiman, 1996] Breiman, L. (1996): Bagging Predictors. Machine Learning (24), pp. 123–140.
- [Dzeroski and Zenko, 2002] Dzeroski S., Zenko B. (2002): Is Combining Classifiers Better than Selecting the Best One?, in Proceedings of the 19th International Conference on Machine Learning, ICML-2002, Morgan Kaufmann Publishers, San Francisco, 2002.
- [Freund and Schapire, 1996] Freund, Y., Schapire R.E. (1996): Experiments with a new boosting algorithm, Proceedings of the International Conference on Machine Learning (ICML-96), pages 148–156, Morgan Kaufmann.
- [Seewald and Fürnkranz, 2001] Seewald A.K., Fürnkranz J. (2001): An Evaluation of Grading Classifiers, in Hoffmann F. et al. (eds.), Advances in Intelligent Data Analysis, 4th International Conference, IDA 2001, Proceedings, Springer, Berlin, pp.115–124, 2001.
- [Seewald, 2002] Seewald A.K. (2002): How to make Stacking Better and Faster While Also Taking Care of an Unknown Weakness, in Proceedings of the 19th International Conference on Machine Learning, ICML-2002, Morgan Kaufmann Publishers, San Francisco, 2002.
- [Seewald, 2003] Seewald A.K. (2003): Towards a Theoretical Framework for Ensemble Classification (extended version), Technical Report, Austrian Research Institute for Artificial Intelligence, Vienna, TR-2003-08.
- [Ting and Witten, 1999] Ting, K. M., Witten, I. H. (1999): Issues in stacked generalization. Journal of Artificial Intelligence Research 10 (1999) 271–289.
- [Wolpert, 1992] Wolpert, D. H. (1992): Stacked generalization. Neural Networks 5(2) (1992) 241–260.

Multiple-Goal Reinforcement Learning with Modular Sarsa(0)

Nathan Sprague

Computer Science Department
University of Rochester
Rochester, NY 14627
sprague@cs.rochester.edu

Dana Ballard

Computer Science Department
University of Rochester
Rochester, NY 14627
dana@cs.rochester.edu

Abstract

We present a new algorithm, GM-Sarsa(0), for finding approximate solutions to multiple-goal reinforcement learning problems that are modeled as composite Markov decision processes. According to our formulation different sub-goals are modeled as MDPs that are coupled by the requirement that they share actions. Existing reinforcement learning algorithms address similar problem formulations by first finding optimal policies for the component MDPs, and then merging these into a policy for the composite task. The problem with such methods is that policies that are optimized separately may or may not perform well when they are merged into a composite solution. Instead of searching for optimal policies for the component MDPs in isolation, our approach finds good policies in the context of the composite task.

keywords: reinforcement learning

1 Introduction

Traditional reinforcement learning algorithms can successfully solve small, single-goal tasks. The main challenge in the area of reinforcement learning is scaling up to larger and more complex problems. The scaling problem takes a number of forms. We may have a problem that has a very large state space, a problem that is best described as a set of hierarchically organized goals and subgoals, or a problem that requires the learning agent to address several tasks at once. It is the last form of scaling that this paper is concerned with.

The naive approach to learning to solve composite tasks is to create a state space that includes all of the information that is relevant to each sub-task. The agent would then learn in this joint space, receiving reward when any of the sub-goals are accomplished. The problem with this approach is that it suffers from the curse of dimensionality; as additional state dimensions are added for each new sub-task, the size of the joint state space grows exponentially.

A more promising approach to this sort of multiple-goal problem is to use the well known Q-learning algorithm to train one learning module to handle each of the sub-goals. The internal Q-values of the different learning modules can

then be used to fairly distribute control among the modules. This approach has been independently explored in [Humphrys, 1996] and [Karlsson, 1997]. It is attractive in its simplicity, and it has shown good empirical performance in a number of domains.

In this paper we will highlight a previously unrecognized problem with existing modular Q-learning algorithms. Existing algorithms learn component policies that may be highly sub-optimal in the context of the composite task, because they do not take into account the fact that the component modules are forced to share control. We will show how to fix this problem by replacing the Q-learning with the closely related Sarsa(0) learning rule. The resulting algorithm shows improved performance on a large sample problem.

2 The problem formalized

The underlying formalism for many reinforcement learning algorithms is the Markov decision process. An MDP, denoted M is described by a 4-tuple (S, A, T, R) , where S is the state space, A is the action space, and $T(s, a, s')$ is the transition function that indicates the probability of arriving in state s' when action a is taken in state s . The reward function $R(s, a)$ denotes the expected one-step payoff for taking action a in state s . The goal of reinforcement learning algorithms is to discover an optimal policy $\pi^*(s)$ that maps states to actions so as to maximize discounted long term reward.

Here we consider the problem of discovering a joint policy for a set of N MDP's $\{M_i\}_1^N$. Throughout we will use subscripts to distinguish the MDPs. These MDP's each have a distinct state space, but they share a common action space, and are required to execute the same action on each time step. This model is intended to map to the case of a single agent that is simultaneously faced with a set of different goals.

The N component MDPs implicitly define a larger composite MDP. Formally, the goal is to find the optimal policy for this composite MDP. The optimal composite policy is defined as the policy that maximizes summed discounted reward across the component MDPs.

The state space of the composite MDP is the cross product of the state spaces of the component MDPs: $S = S_1 \times S_2 \times \dots \times S_N$. The composite reward function is defined as: $R(s, a) = \sum_{i=1}^N R_i(s_i, a)$. In the case where the component MDPs are independent, the composite transition function can

be written as: $T(s, a, s') = \prod_{i=1}^N T_i(s_i, a, s'_i)$. In the case where the component MDPs are not independent, the exact composite transition function will depend on the particular dependencies between the models.

In theory, there is no reason that the composite MDP could not be solved directly using the traditional Q-learning algorithm. However, this is generally not practical because the size of the composite state space may grow exponentially with the number of component MDPs.

3 Modular Q-Learning

Humphrys and Karlsson [Humphrys, 1996; Karlsson, 1997] independently developed similar approaches to the problem of multiple-goal reinforcement learning. The idea is that a separate learning module is created for each component MDP. The agent takes actions in the environment, and each module i is trained with the standard Q-learning update rule:

$$Q_i(s_i, a) \leftarrow (1-\alpha)Q_i(s_i, a) + \alpha(r_i + \gamma \max_{a'} Q_i(s'_i, a')) \quad (1)$$

Where r_i is the immediate reward, α is the learning rate parameter, and γ is a discount factor applied to future rewards. In single goal reinforcement learning problems, these Q-values are used only to rank order the actions in a given state. The key observation here is that the Q-values can also be used in multiple-goal problems to indicate the degree of preference that modules have for different actions. There are several possible ways these values can be used to select a compromise action to execute. The different approaches will be referred to as action selection mechanisms.

Karlsson's suggestion, which he calls "greatest mass", is to generate an overall Q-value as a simple sum of the Q-values of the individual modules: $Q(s, a) = \sum_{i=1}^n Q_i(s_i, a)$. The action with the maximum summed value is then chosen to execute. We will refer to this approach as GM-Q for greatest mass Q-learning.

Humphrys considers the greatest mass approach, but raises the objection that the action with the highest sum may not be particularly good for any of the modules, with the result that no module is able to reach its goal. He explores several winner-take-all alternatives that constrain the chosen action to be optimal for at least one module. For a given state s each of the N modules promotes its own action with a value $W_i(s_i)$. The module with the largest W value is then allowed to execute its preferred action.

The simplest method for generating the W -values, which we will refer to as Top-Q, is to set $W_i(s_i) = \max_a Q_i(s_i, a)$, thus giving control to the module with the highest Q-value in the current state. This method suffers from the drawback that the module with the highest Q-value may have no preference over what action is chosen, while another module stands to lose a great deal if its action is not selected. The method sometimes exhibits reasonable performance, but this is strongly dependent on the structure of the reward functions.

A better alternative, referred to as negotiated W-learning, is to grant control to the module that stands to lose the most long term reward if it is not selected. This module can be discovered by examining the Q-values for the current state. Refer to [Humphrys, 1996] for a detailed description of the algorithm.

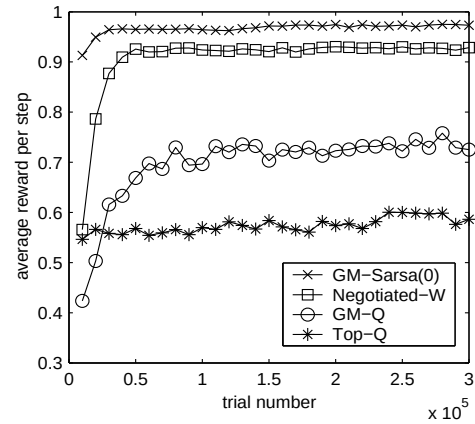


Figure 1: Results of one training run for GM-Sarsa(0) and three Q-learning based algorithms on a food gathering task. Training is divided into trials lasting 100 time steps. Data points are generated by suspending training every 10000 trials and computing the mean performance for 1000 trials without exploration. Each algorithm uses an ϵ -greedy exploration policy with ϵ linearly reduced from .4 to 0 during the first half of trials. All algorithms use a fixed learning rate of .05, and a discount factor of .9.

3.1 The problem with modular Q-learning

Q-learning has some attractive qualities as a basis for multiple-goal reinforcement learning. Chief of these is the fact that it is an off-policy learning method. This means that Q-learning for a single MDP is guaranteed to converge to the optimal solution regardless of what policy is followed during training, as long as each state-action pair is visited infinitely often in the limit. This fact makes it easy to prove convergence results for the composite reinforcement learning algorithms introduced above. In particular, it is easy to see that each module is guaranteed to converge to the optimal policy and value function for its own MDP. Since the action selection mechanisms generate a policy deterministically from the component value functions, the composite policy is also guaranteed to converge, although there is no guarantee concerning the quality of the composite solution.

Unfortunately, the off-policy character of Q-learning is also a serious limitation. The difficulty is that the one-step value updates for each module are computed under the assumption that all future actions will be chosen optimally for that MDP. This assumption is not valid under the action selection mechanisms described above; future actions will represent some compromise policy in which the different modules share control. This means that the computed Q-values do not converge to the actual expected return under the composite policy. Instead, the $\max$ in equation (1) results in Q-values with a positive bias.

4 Modular Sarsa(0)

A possible solution to the problem of positive bias is to replace Q-learning with an on-policy learning algorithm. In particular we will explore the use of Sarsa(0) [Rummery and

Niranjan, 1994; Singh and Sutton, 1996; Sutton, 1996]. The update rule for Sarsa(0) is:

$$Q_i(s_i, a) \leftarrow (1 - \alpha)Q_i(s_i, a) + \alpha(r_i + \gamma Q_i(s'_i, a')) \quad (2)$$

This update rule is virtually identical to that for Q-learning except that the max over Q-values on the right has been replaced with the Q-value of the state action pair that is actually observed on the next step. For the case of single MDPs Sarsa(0) has been proved to converge to the optimal policy as long as the exploration rate is asymptotically decayed toward zero according to an appropriate schedule [Singh *et al.*, 2000].

The key observation for our purposes is that, since Sarsa(0) is an on-policy method, it does not suffer from the problem of positive bias. Since updates are based on the actions that are actually taken, rather than on the best possible action, we expect Sarsa(0) based modules to discover Q-values that are closer to the true expected return under the composite policy.

Any of the action selection mechanisms from Section 3 could be recast to use Sarsa(0) rather than Q-learning to train the modules. However, we focus on the method of greatest mass. We refer to the resulting algorithm as GM-Sarsa(0). Recall that the goal is to maximize the summed reward across all of the component MDPs. Assuming that we have trustworthy utility estimates from each of the modules, it makes sense to choose the action that has the highest summed utility across all of the modules. By definition, this is the action that will lead to the greatest summed long term reward. This reasoning did not hold under Q-learning, because the utility estimates were inaccurate under the composite policy.

Thus far we have no convergence proof for the GM-Sarsa(0) algorithm. Refer to the associated technical report [Sprague and Ballard, 2003] for a discussion of the possible convergence characteristics.

5 Examples

Figure 1 demonstrates the performance of GM-Sarsa(0) on a sample composite task (the task is adapted from [Singh and Cohn, 1998]). The goal of the agent in this task is to gather stationary food items while avoiding a predator in a 5×5 grid. There are three food items present at all times.

The agent moves in any of the eight possible directions at each time step. A random move is made with a probability of .1. If the agent contacts any of the food items it receives a reward of 1.0, and the item is randomly moved to a new position. The agent receives a reward of .5 for every time step that it avoids the predator. The predator moves deterministically one position toward the agent on every other time step.

The positions of the food items as well as the positions of the agent and predator result in $25^5 \approx 10$ million distinct states. This is too large for a monolithic tabular learning algorithm to be practical. The task is a good candidate for a modular reinforcement learning algorithm because it can be decomposed into several small MDPs. One MDP describes the agent's interaction with the predator, and three MDPs describe the interaction with the food items. Each of these component MDPs has $25^2 = 625$ states. Figure 1 shows the performance of GM-Sarsa(0) as well as the three Q-learning

based algorithms from Section 3 on this task. Of the four algorithms, GM-Sarsa(0) exhibits the best performance.

6 Conclusion

We have presented a method for learning approximately optimal policies for a certain class of composite Markov decision processes. Empirical results demonstrate that our approach performs better than a number of existing algorithms. Future work will focus on proving convergence results for our algorithm. A longer version of this paper, including a discussion of related work is available as [Sprague and Ballard, 2003].

Acknowledgments

This material is based upon work supported by a grant from the Department of Education under grant number P200A000306, a grant from the National Institutes of Health under grant number 5P41RR09283 and a grant from the National Science Foundation under grant number E1A-0080124. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the above mentioned institutions.

References

- [Humphrys, 1996] M. Humphrys. Action selection methods using reinforcement learning. In *From Animals to Animals 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 135–144, Cambridge, MA, 1996.
- [Karlsson, 1997] J. Karlsson. *Learning to Solve Multiple Goals*. PhD thesis, University of Rochester, 1997.
- [Rummery and Niranjan, 1994] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.
- [Singh and Cohn, 1998] S. Singh and D. Cohn. How to dynamically merge markov decision processes. In *Advances in Neural Information Processing Systems*, volume 10, 1998.
- [Singh and Sutton, 1996] S. Singh and R. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3), 1996.
- [Singh *et al.*, 2000] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvari. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 2000.
- [Sprague and Ballard, 2003] N. Sprague and D. Ballard. Multiple-goal reinforcement learning with modular sarsa(0). Technical Report 798, University of Rochester Computer Science Department, 2003.
- [Sutton, 1996] R. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, volume 8, 1996.

Integrating Background Knowledge Into Text Classification

Sarah Zelikovitz

College of Staten Island – CUNY
2800 Victory Blvd
Staten Island, NY 11694
zelikovitz@postbox.csi.cuny.edu

Haym Hirsh

Rutgers University
110 Frelinghuysen Road
Piscataway, NJ 08854
hirsh@cs.rutgers.edu

Abstract

We present a description of three different algorithms that use background knowledge to improve text classifiers. One uses the background knowledge as an index into the set of training examples. The second method uses background knowledge to reexpress the training examples. The last method treats pieces of background knowledge as unlabeled examples, and actually classifies them. The choice of background knowledge affects each method's performance and we discuss which type of background knowledge is most useful for each specific method.

1 Using Background Knowledge

Supervised learning algorithms rely on a corpus of labeled training examples to produce accurate automatic text classifiers. An insufficient number of training examples often results in learned models that are suboptimal when classifying previously unseen examples. Numerous different approaches have been taken to compensate for the lack of training examples. These include the use of unlabeled examples [Bennet and Demiriz, 1998; Blum and Mitchell, 1998; Nigam *et al.*, 2000; Goldman and Zhou, 2000], the use of test examples [Joachims, 1999], and choosing a small set of specific unlabeled examples to be manually classified [Lewis and Gale, 1994].

Our approach does not assume the availability of either unlabeled examples or test examples. As a result of the explosion of the amount of data that is available, it is often the case that text, databases and other sources of knowledge that are related to the text classification task are readily available from the World Wide Web. We incorporate such “background knowledge” into different learners to improve classification of unknown instances. The use of external readily available textual resources allows learning systems to model the domain in a way that would be impossible by simply using a small set of training instances. For example, if a text classification task is to determine the sub-discipline of physics that a paper *title* should belong to, background knowledge such as abstracts, physics newsgroups, and perhaps even book reviews of physics books can be used by learners to create more accurate classifiers.

We present three methods of incorporating background knowledge into the text classification task. Each of these methods uses the corpus of background knowledge in a different way, yet empirically, on a wide variety of text classification tasks we can show that accuracy on test sets can be improved when incorporating background knowledge into these systems. We ran all three methods incorporating background knowledge on a range of problems from nine different text classification tasks. Details on the data sets can be found at (www.cs.csi.cuny.edu/~zelikovi/datasets; each varied on the size of each example, the size of each piece of background knowledge, the number of examples and number of items of background knowledge, and the relationship of the background knowledge to the classification task.

2 Methods

In our first approach we use Naive Bayes and EM as in [Nigam *et al.*, 2000]. We can substitute more general background knowledge for unlabeled examples, and obtain improvements in accuracy on text classifiers that are created using both the training set and the set of background knowledge. Naive Bayes classifiers make the assumption that examples (both labeled and unlabeled) have been generated by a mixture model that has a one-to-one correspondence with classes. Even if this assumption is true for the labeled data and the test data, by its very nature, background knowledge should not fit this assumption at all. However, the interesting observation that we make is that to gain leverage out of unlabeled examples, the unlabeled data that we have need not be specifically and accurately unlabeled examples. As long as the vocabulary and classification structure closely resembles the training/test data, background knowledge can improve classification accuracy in textual data using the EM algorithm.

A second approach that we take is based upon a nearest neighbor text classifier using WHIRL [Cohen, 1998; Cohen and Hirsh, 1998]. Instead of simply comparing a test example to the corpus of training examples, we use the items of background knowledge as “bridges” to connect each new example with labeled training examples. A labeled training example is useful in classifying an unknown test instance if there exists some set of unlabeled background knowledge that is similar to both the test example and the training example. We call this a “second-order” approach to classification [Zelikovitz and Hirsh, 2000; 2002], in that data are no longer di-

rectly compared but rather, are compared one step removed, through an intermediary.

Finally we use the background knowledge to redescribe both the training and the test examples. To do this, we add the background knowledge documents to the training set, to create a large, sparse term-by-document ($t \times d$) matrix. We then use Latent Semantic Indexing (LSI) [Deerwester *et al.*, 1990] to automatically redescribe textual data in a new smaller semantic space using singular value decomposition. The original space is decomposed into linearly independent dimensions or “factors”, and the terms and documents of the training and test examples are then represented in this new vector space [Zelikovitz and Hirsh, 2001; 2002]. Documents with high similarity no longer simply share words with each other, but instead are located near each other in the new semantic space. Since this semantic space was created by incorporating the background knowledge, the model of the domain that it creates reflects both the training set and the background knowledge.

3 Comparison of Approaches

Different types of background knowledge are most useful for each of these three systems. The system based upon WHIRL performs best on the problems where the form and size of the background knowledge is substantially different than the training and test data. For example, we classify names of companies by area using Yahoo! pages as background knowledge. These background pieces of data are not really classifiable, in the sense that they do not necessarily belong to any specific class. Since this WHIRL-based method does not attempt to classify the background knowledge, but merely uses it to index into the training corpus, it makes the best use of this background knowledge.

For the data sets where the background knowledge fits very closely to the training and test classification task, EM outperforms the other systems. For example, EM performed best when classifying physics papers by subdiscipline using abstracts as background knowledge. This is consistent with the way EM makes use of background knowledge. Since EM actually classifies the background knowledge, and uses the background knowledge to decide on the parameters of its generative model, the closer the background knowledge is to the training and test sets, the better EM will perform. Ideally, for EM, we wish the background knowledge to be generated from the same model as the training and test sets.

Reexpressing the data and background with LSI seems to be most effective when there is very limited training data. On the smallest data sets, it outperforms all the other methods in many domains. When very few training examples exist, this method can still build a space that correctly models the domain by using the available background knowledge.

We are currently looking at methods to evaluate sets of background knowledge to determine the amount of background knowledge as well as the measure of relevance that it must have to the training set to be useful for each of these learners.

References

- [Bennet and Demiriz, 1998] K. Bennet and A. Demiriz. Semi-supervised support vector machines. *Advances in Neural Information Processing Systems*, 12:368–374, 1998.
- [Blum and Mitchell, 1998] A. Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 92–100, 1998.
- [Cohen and Hirsh, 1998] William Cohen and Haym Hirsh. Joins that generalize: Text categorization using WHIRL. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 169–173, 1998.
- [Cohen, 1998] William Cohen. Integration on heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of ACM-SIGMOD 98*, 1998.
- [Deerwester *et al.*, 1990] S. Deerwester, S. Dumais, G. Furnas, and T. Landauer. Indexing by latent semantic analysis. *Journal for the American Society for Information Science*, 41(6):391–407, 1990.
- [Goldman and Zhou, 2000] S. Goldman and Y. Zhou. Enhancing supervised learning with unlabeled data. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.
- [Joachims, 1999] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 200–209, 1999.
- [Lewis and Gale, 1994] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *SIGIR94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, 1994.
- [Nigam *et al.*, 2000] Kamal Nigam, Andre Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.
- [Zelikovitz and Hirsh, 2000] S. Zelikovitz and H. Hirsh. Improving short text classification using unlabeled background knowledge to assess document similarity. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 1183–1190, 2000.
- [Zelikovitz and Hirsh, 2001] S. Zelikovitz and H. Hirsh. Using LSI for text classification in the presence of background text. In *Proceedings of the Tenth Conference for Information and Knowledge Management*, 2001.
- [Zelikovitz and Hirsh, 2002] S. Zelikovitz and H. Hirsh. Integrating background knowledge into nearest-Neighbor text classification. In *Proceedings of the 6th European Conference on Case Based Reasoning*, 2002.

Parametric Distance Metric Learning with Label Information*

Zhihua Zhang, James T. Kwok and Dit-Yan Yeung

Hong Kong University of Science and Technology

Clear Water Bay, Kowloon, Hong Kong

{zhzhang, jamesk, dyyeung}@cs.ust.hk

Abstract

Distance-based methods in pattern recognition and machine learning have to rely on a similarity or dissimilarity measure between patterns in the input space. For many applications, Euclidean distance in the input space is not a good choice and hence more complicated distance metrics have to be used. In this paper, we propose a parametric method for metric learning based on class label information. We first define a dissimilarity measure that can be proved to be metric. It has the favorable property that between-class dissimilarity is always larger than within-class dissimilarity. We then perform parametric learning to find a regression mapping from the input space to a feature space, such that the dissimilarity between patterns in the input space is approximated by the Euclidean distance between points in the feature space. Parametric learning is performed using the iterative majorization algorithm. Experimental results on real-world benchmark data sets show that this approach is promising.

1 Introduction

The notion of similarity or dissimilarity plays a fundamental role in pattern recognition and machine learning. A promising direction to pursue is to learn good (dis)similarity measures from data. Recently, learning distance metrics from data has aroused a great deal of interest from machine learning researchers. One typically wants to embed patterns in a (possibly non-metric) input space into a feature space, in which the Euclidean distance between points accurately reflects the dissimilarity between the corresponding patterns. Therefore the (linear or nonlinear) mapping from the input space to the feature space corresponds to feature extraction. Alternatively, the feature space may be a low-dimensional space for data visualization.

In this paper, we propose a parametric distance metric learning method in the supervised setting. The main ideas

of our method are summarized as follows. Using class label information, we define a similarity measure (and hence also the corresponding dissimilarity measure) between patterns in the input space. The dissimilarity measure implicitly induces a metric space for embedding the original patterns. To explicitly represent the mapping from the input space to the feature space, we then approximate the mapping by a regression model to embed the original patterns in an Euclidean space. The regression parameters are estimated from data with the objective that the dissimilarity between patterns in the input space is approximated by the Euclidean distance between points in the feature space. Once the regression model has been found, any new pattern can be mapped to its corresponding location in the feature space. Distance-based methods, such as k -means clustering, nearest neighbor classifiers and support vector machines, can then be applied in the feature space for clustering or classification applications.

The rest of this paper is organized as follows. A modified metric incorporating class label information is proposed in Section 2. Section 3 outlines our regression model for metric learning and the corresponding optimization method. Experimental results are presented in Section 4, and the last section gives some concluding remarks.

2 Modified Metric with Label Information

Denote the input space by $\mathbb{R}^q$ and the set of all C possible class (target) labels by $\mathcal{T}$. A training set $\mathcal{D} \subseteq \mathbb{R}^q \times \mathcal{T}$ has n patterns $\{(\mathbf{x}_1, t_1), \dots, (\mathbf{x}_n, t_n)\}$, where $t_i = r$ if pattern i belongs to class r . Here, each pattern is assumed to belong to only one class. In general, a number of similarity measures can be defined on these patterns [Gower and Legendre, 1986]. In this paper, we utilize also the label information in defining the similarity s_{ij} between patterns $\mathbf{x}_i$ and $\mathbf{x}_j$:

$$s_{ij} = \begin{cases} \frac{1}{2} + \frac{1}{2} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\beta}\right) & t_i = t_j \\ \frac{1}{2} - \frac{1}{2} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\beta}\right) & t_i \neq t_j \end{cases}, \quad (1)$$

where $\|\cdot\|$ denotes the Euclidean norm and $\beta > 0$ is a width parameter. The corresponding dissimilarity δ_{ij} is then:

$$\begin{aligned} \delta_{ij} &= s_{ii} + s_{jj} - 2s_{ij} \\ &= \begin{cases} 1 - \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\beta}\right) & t_i = t_j \\ 1 + \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\beta}\right) & t_i \neq t_j \end{cases}. \end{aligned} \quad (2)$$

*This research has been partially supported by the Research Grants Council of the Hong Kong Special Administrative Region under grants DAG01/02.EG28, HKUST2033/00E and HKUST6195/02E.

As illustrated in Figure 1, this (dis)similarity measure enjoys some nice properties for pattern discrimination. For example, the (dis)similarity between any two patterns in the same class is always larger (smaller) than that between any two patterns belonging to different classes. Moreover, the larger the Euclidean distance between the patterns is, the smaller is the within-class similarity while the larger is the between-class similarity.

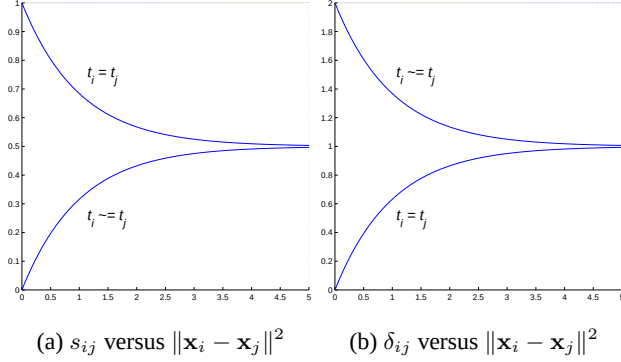


Figure 1: Similarity and dissimilarity in (1) and (2).

In recent years, finite metric spaces and their embeddings have received much attention [Indyk, 2001; Linial *et al.*, 1995]. Among embedding into normed spaces, embedding into an Euclidean space is the most popular.

Given the dissimilarity matrix $\Delta = [\delta_{ij}]_{n \times n}$, we are interested in the question of whether and how the dissimilarity matrix Δ can be embedded. In other words, for the original points $\{x_1, \dots, x_n\}$, we attempt to find a configuration of points $\{\hat{x}_1, \dots, \hat{x}_n\}$ in some Euclidean space $\mathbb{R}^l$ such that the squared distances between these points will be equal to the so-defined dissimilarities δ_{ij} .

The following theorem confirms that Δ can be embedded.

Theorem 1 Define $d_{ij} = \sqrt{\delta_{ij}}$, with δ_{ij} as in (2). The matrix $D = [d_{ij}]$ is metric. In other words, d_{ij} satisfies the following properties:

1. $d_{ij} \geq 0$, $\forall i, j$,
2. $d_{ii} = 0$, $\forall i$,
3. $d_{ij} = d_{ji}$, $\forall i, j$,
4. $d_{ik} + d_{jk} \geq d_{ij}$, $\forall i, j, k$.

The proof of this theorem can be found in [Zhang *et al.*, 2003]. The subsequent task is then to find the embedding, i.e., points $\{\hat{x}_1, \dots, \hat{x}_n\} \in \mathbb{R}^l$ such that the inter-point distance is equal to d_{ij} . In general, obtaining an exact solution for these $\hat{x}_i$'s is difficult. Nevertheless, because D is metric, an approximate solution can be easily obtained by using principal coordinate analysis or other multidimensional scaling (MDS) methods [Cox and Cox, 2000]. We will return to this problem in Section 3.

Notice that the resultant Euclidean embedding will still incorporate information from both the input space representation (x_i) of the patterns and their corresponding class labels

(t_i). Moreover, the distance metric d_{ij} , like the associated δ_{ij} , enjoys those nice properties useful for pattern discrimination.

3 Metric Learning with Regression Model

As mentioned in Section 2, an approximate solution for $\{\hat{x}_1, \dots, \hat{x}_n\}$ can be obtained by using MDS. However, this may be intractable for large data sets. Moreover, for new patterns with unknown labels, the problem then is on how to determine s_{ij} in the first place. Following [Koontz and Fukunaga, 1972; Cox and Ferry, 1993; Webb, 1995], we attempt to find a mapping from x_i in the original input space $\mathbb{R}^q$ to $\hat{x}_i$ in the embedded Euclidean space $\mathbb{R}^l$. One possibility is to first obtain a MDS configuration, and then construct a regression model from x_i to $\hat{x}_i$ [Cox and Ferry, 1993]. However, this mapping is not determined as part of the MDS procedure [Webb, 1995]. In the following, we will follow the approach of [Webb, 1995].

Denote the mapping from the original input space $\mathbb{R}^q$ to the embedded Euclidean space $\mathbb{R}^l$ by $f = (f_1, \dots, f_l)'$. Assume that each f_i is a linear combination of p basis functions:

$$f_i(x; \mathbf{W}) = \sum_{j=1}^p w_{ji} \phi_j(x), \quad (3)$$

where $\mathbf{W} = [w_{ji}]_{p \times l}$ contains the free parameters, and the $\phi_j(x)$'s are basis functions that can be linear or nonlinear. The regression mapping (3) can be written in matrix form as

$$\mathbf{y}(\mathbf{W}) = \mathbf{f}(x; \mathbf{W}) = \mathbf{W}' \phi(x),$$

where $\phi(x) = [\phi_1(x), \dots, \phi_p(x)]'$. Let $\mathbf{X}$ be the target configuration, with $d_{ij}(\mathbf{X}) = \sqrt{\delta_{ij}}$ where δ_{ij} is defined in (2). Using the iterative majorization algorithm, we then minimize the squared error

$$e^2(\mathbf{W}) = \sum_{i=1}^n \sum_{j=1}^n (d_{ij}(\mathbf{X}) - q_{ij}(\mathbf{W}))^2 \quad (4)$$

w.r.t. $\mathbf{W}$, where $q_{ij}(\mathbf{W}) = \|\mathbf{W}'(\phi(x_i) - \phi(x_j))\|$. More details can be found in [Zhang *et al.*, 2003].

4 Experiments

In this Section, we perform experiments on six benchmark data sets (Table 1) from the UCI repository [Murphy and Aha, 1994]. The distance metric is learned using a small subset of the labeled patterns, with $l = p = q$, $\phi(x) = x$ and the width β in (1) set to the average distance of the labeled patterns to the class means. The remaining patterns are then used for testing.

Table 2 shows the classification results by the nearest mean and nearest neighbor classifiers, with both the Euclidean and learned metrics. As can be seen, the learned metric almost always outperforms the original metric.

Next, we perform clustering experiments using the k -means clustering algorithm, with the value of k set to the true number of clusters in each data set. The clustered patterns are assigned labels and the clustering accuracy is measured by comparing these labels with the true labels (as in classification problems). As these cluster labels can be permuted

Table 1: The six UCI data sets used in the experiments.

data set	total # of patterns		# patterns for metric learning
	class	size	
Pima Indians	1	500	80
diabetes (diabetes)	2	268	50
soybean	1	10	2
	2	10	2
	3	10	2
	4	17	4
wine	1	59	20
	2	71	20
	3	48	20
Wisconsin breast cancer (WBC)	1	212	50
	2	357	50
ionosphere	1	126	50
	2	225	50
iris	1	50	10
	2	50	10
	3	50	10

without changing the clustering solution, results reported here are based on the labeling with the highest clustering accuracy. As can be seen from Table 3, the learned metric outperforms that with the original metric on all data sets.

Table 2: Classification accuracies on the UCI data sets (Numbers in bold indicate the better results).

data set	nearest mean		nearest neighbor	
	Euclidean metric	learned metric	Euclidean metric	learned metric
diabetes	463/638	475/638	432/638	425/638
soybean	36/37	37/37	35/37	37/37
wine	86/118	115/118	77/118	117/118
WBC	430/469	451/469	420/469	453/469
ionosphere	159/251	201/251	212/251	225/251
iris	108/120	110/120	114/120	114/120

5 Concluding Remarks

In this paper, we proposed a new parametric method for distance metric learning based on class label information. Experiments on UCI data sets show promising results.

The current work can be extended in several directions. First, nonlinear basis functions can be used to improve the approximation power of the regression mapping. Second, although Theorem 1 states that the dissimilarity measure induces a metric, it is not clear whether the matrix is also Euclidean. If this is the case, a new kernel can then be defined on the joint space of the input space and class label space [Schölkopf, 2002]. Third, in addition to using label information, we will also incorporate manifold structure between neighboring patterns into our metric learning process.

Table 3: Clustering accuracies on the UCI data sets (Numbers in bold indicate the better results).

data set	Euclidean metric	learned metric
diabetes	459/638	480/638
soybean	37/37	37/37
wine	85/118	117/118
WBC	412/469	446/469
ionosphere	168/251	221/251
iris	107/120	110/120

References

- [Cox and Cox, 2000] T.F. Cox and M.A.A. Cox. *Multidimensional Scaling*. Chapman & Hall/CRC, second edition, 2000.
- [Cox and Ferry, 1993] T.F. Cox and G. Ferry. Discriminant analysis using non-metric multidimensional scaling. *Pattern Recognition*, 26(1):145–153, 1993.
- [Gower and Legendre, 1986] J.C. Gower and P. Legendre. Metric and Euclidean properties of dissimilarities coefficients. *Journal of Classification*, 3:5–48, 1986.
- [Indyk, 2001] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 10–33, 2001.
- [Koontz and Fukunaga, 1972] W.L.G. Koontz and K. Fukunaga. A nonlinear feature extraction algorithm using distance information. *IEEE Transactions on Computers*, 21(1):56–63, 1972.
- [Linial et al., 1995] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.
- [Murphy and Aha, 1994] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases. <http://www.ics.uci.edu/~mlearn/MLRepository.html>, 1994.
- [Schölkopf, 2002] B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, 2002.
- [Webb, 1995] A.R. Webb. Multidimensional scaling by iterative majorization using radial basis functions. *Pattern Recognition*, 28(5):753–759, 1995.
- [Zhang et al., 2003] Z. Zhang, J.T. Kwok and D.Y. Yeung. Parametric distance metric with label information. <ftp://ftp.cs.ust.hk/pub/techreport/03/tr03-02.ps.gz>, 2003.