Machine Learning, 13, 7–33 (1993)

Cost-Sensitive Learning of Classification Knowledge and Its Applications in Robotics

MING TAN TAN@GTE.COM
GTE Laboratories Incorporated, 40 Sylvan Road, Waltham, MA 02254

Editor: Steve Minton

Abstract. Traditional learning-from-examples methods assume that examples are given beforehand and all features are measured for each example. However, in many robotic domains the number of features that could be measured is very large, the cost of measuring those features is significant, and thus the robot must judiciously select which features it will measure. Finding a proper tradeoff between the *accuracy* (e.g., number of prediction errors) and *efficiency* (e.g., cost of measuring features) during learning (prior to convergence) is an important part of the problem. Inspired by such robotic domains, this article considers realistic measurement costs of features in the process of incremental learning of classification knowledge. It proposes a unified framework for learning-from-examples methods that trade off accuracy for efficiency during learning, and analyzes two methods (CS-ID3 and CS-IBL) in detail. Moreover, this article illustrates the application of such a cost-sensitive-learning method to a real robot designed for an *approach-recognize* task. The resulting robot learns to approach, recognize, and grasp objects on a floor effectively and efficiently. Experimental results show that highly accurate classification procedures can be learned without sacrificing efficiency in the case of both synthetic and real domains.

Keywords. cost-sensitive learning, robot learning, classification, active perception, sensing cost

1. Introduction

Consider a simple mobile robot whose primary goal is to pick up various objects on a floor and discard them. At some point the robot must determine which of its sensing procedures allow it to distinguish among different grasping actions. In the general situation, a robot agent must decide which of its available information resources are actually useful. A classification procedure is such a measurement policy that can be used to classify an object. Although classification procedures have been used extensively in areas of robotics for pattern recognition, object manipulation, and navigation, their effective and efficient application is by no means easy. First, if a robot operates in a changing environment, its classification procedures must be updated incrementally. Second, during classification, sensing too much of an object wastes valuable resources, while sensing too little yields an insufficient description. Third, actions embedded in a classification procedure must be executed efficiently. Fourth, a robot perceives an object through its sensors, so its mental picture of that object is likely to be distorted because sensors are often inaccurate. These factors not only make it difficult to program classification procedures, but also undermine purely theoretical or model-based approaches (Mason, 1985; Stansfield, 1989; Trinkle & Paul, 1989). In constrast, a learning approach allows a robot to empirically adapt to the particular world it is in. This avoids the complexity of formal methods and minimizes dependence

on human intervention. Robot learning techniques can improve the connection of a robot's perception to action.

However, the majority of work in robot learning has not dealt realistically with costs associated with perception and action. A common assumption is that a robot is provided a complete description of the external world at no cost and that its sensing procedures and actions can be executed instantaneously. For example, an object-picking robot can use learning-from-examples methods to learn classification procedures, by mapping objects to examples, sensing procedures to features, and grasping actions to classes of examples. But traditional learning-from-examples methods often assume that all features are measured for each example and that examples are given beforehand. In reality, robots only have limited resources, and their sensing procedures may incur significant expense, limiting their abilities to routinely execute a large number of sensing procedures to gather information. Furthermore, robots usually encounter objects sequentially (e.g., an object-picking robot may not see discarded objects again) and they must gather information and learn incrementally. Since most of the information in the external world is irrelevant to the immediate task (e.g., classification) faced by a robot and only interferes with learning, a learning robot must be selective in its information gathering and able to learn from incomplete information. Particularly, a learning-from-examples robot must decide where to sense and which sensing features to measure in order not only to classify each object efficiently but also to generate (incomplete) examples efficiently for incremental learning. Consequently, there is a tradeoff between *accuracy* (e.g., number of prediction errors) and *efficiency* (e.g., average cost of measuring features per object) during such robot learning. Finding the proper tradeoff is essential for robot learning.

Accuracy and efficiency during learning are two important criteria for robot learning methods. Accuracy can be measured in terms of execution outcome (e.g., whether a robot picked up an object or not). Efficiency can be measured in terms of execution cost (e.g., moving and sensing time) or computational cost. This article mainly focuses on the execution cost (assuming the computational cost is negligible); if necessary, the computational cost can be handled similarly. For a robot that incrementally learns a classification procedure from a sequence of objects, its cost for each object arise from both classifying an object and gathering additional object information (e.g., when a predication is incorrect). Only a few researchers like Nunez (1991) and Norton (1989) investigated the role of non-computational costs in learning classification knowledge. Although their specific methods achieve asymptotic performances (after convergence) similar to one of ours, their methods assume that complete examples are given first and are non-incremental. This limits their applications in such robotic domains where the number of features is very large and a robot agent cannot afford to measure every feature. In fact, the tradeoff between cumulative classification accuracy and average cost per object during learning has never been studied.

This article is a study of cost-sensitive learning of classification procedures and its applications in robotics. It investigates the impact of execution costs on learning approaches and addresses the issue of active perception. It proposes a novel framework for cost-sensitive learning-from-examples methods that trade off accuracy for efficiency during learning, and analyzes two methods in detail. Moreover, this article illustrates the application of such a cost-sensitive-learning method to a real robot designed for an *approach-recognize* task. The main thesis is that a learning robot should focus on both accuracy and efficiency during

learning, and it is possible to gradually improve the former without sacrificing the latter on a per object basis.

Specifically, section 2 presents a unified framework for cost-sensitive learning classification knowledge. Within this framework, various cost-sensitive-learning methods can be instantiated: they all learn classification knowledge from examples and also construct examples incrementally from scratch. Two such cost-sensitive-learning methods, i.e., a decision-tree-based method (CS-ID3) and an instance-based method (CS-IBL), are developed, and their tradeoffs between accuracy and efficiency during learning in a synthetic domain are evaluated.

Section 3 then describes an application of the decision-tree-based cost-sensitive-learning method to an approach-recognize task for a real robot. Given a sequence of unknown objects and the execution times of sensors and actions, a cost-sensitive-learning robot called CS-HERO learns where to sense, which sensor to use, and which action to apply. As a result, it learns to approach, recognize, and grasp objects efficiently. Compared with traditional learning approaches, in experiments involving 14 training objects and 11 test objects, CS-HERO is an order of magnitude faster during training and at least 30% faster during testing. Its overall success rate on novel objects is over 80%, and many failures can be recovered later through incremental learning.

Finally, section 4 compares cost-sensitive-learning methods with other learning techniques, and section 5 points out limitations and draws some conclusions.

2. Cost-sensitive learning

Cost-sensitive learning (*CS-learning*) is a new inductive technique that incrementally acquires a classification procedure from self-constructed examples and seeks to minimize the average measurement cost per object. Specifically, given 1) a sequence of unknown objects and 2) a set of features whose values for each object can be measured at known costs, a CS-learning method incrementally learns a class description from saved examples, uses the description to predict the class of a new object, and saves the new (maybe incomplete) example if necessary. A CS-learning method repeats this process for each object. At the same time, it attempts to reduce the overall cost of measuring features during both construction and use of the class description. We assume that the verification of the classes of objects can be either provided by an outside agent or determined by a learning agent's own experimentation. The order of objects can be arbitrary, and each object, once encountered, may or may not be seen again.

Assume that each example is represented as a set of feature-value pairs and a class lable. Table 1 outlines a unified framework for CS-learning methods. For each new object, the framework first identifies a set of saved examples consistent with already measured feature values of the new object. Unless it can predict a class from these identified examples, the framework selects a potential cost-effective feature from them. It then measures the selected feature and records the value in the new example (initially empty). This cycle is repeated until there is a prediction. If the predicted class differs from the correct class, a prediction error occurs. In this case, the framework will repeatedly measure additional features of the new object to expand the new example until the new example can be discriminated

Table 1. The CS-learning framework.

For each new object:

1. Retrieve all saved examples from a library.
2. Repeat
   (a) **Identify** a set of saved examples consistent with already measured values of the new object.
   (b) If a class can be **predicted** from the identified examples, go to 3.
   (c) **Select** a new feature that has potential to be cost-effective from identified examples.
   (d) Measure the selected feature and record its value in the new example (initially empty).
3. Verify the predicted class from environment.
4. If the predicted class is incorrect, **discriminate** the new example from conflicting examples by measuring additional potentially cost-effective features for the new object.
5. **Update** the example library to reflect the new example.

from the conflicting examples that contributed to the prediction error. A *conflicting example* is an example that has a different class but has consistent feature values (and possibly more). Finally, it updates the example library to reflect the new example that has measured features and the correct class. The framework has five generic functions, highlighted in table 1:

1. **Example identification**—Determine the relevance of examples, i.e., which examples are to be identified at each stage of selection. In general, a distance between a saved example and a new example (e.g., Euclidean distance:

$$\sqrt{(fv_1^{saved} - fv_1^{new})^2 + (fv_2^{saved} - fv_2^{new})^2 + \ldots}$$

   where $fv_i^{saved/new}$ is the value of the ith feature of a saved/new example) can be used to define such relevance quantitatively. Normally, numeric features are normalized to the range [0, 1]; non-numeric features have a difference of 0 if equal in value and 1 otherwise. If one of two compared feature values is unknown, it is assigned a difference 0.5. Using such a definition, a CS-learning method can focus its attention on the examples whose distances differ from the distance of the current best matched example(s) by no more than a parameter d. The greater d is, the more examples as identified, and the less focused a CS-learning method will be. If d is equal to 0, only the best matched examples are identified.
2. **Class prediction**—Predict a class from identified examples using sufficiency and/or necessity criteria. For example, if all the remaining identified examples have the same class, predict the class.
3. **Feature selection**—Select a feature from identified examples to discriminate them (e.g., using hill-climbing heuristics or a look-ahead search that explores the dependency among features). The measurement costs of features are taken into account during the selection.
4. **Example discrimination**—Measure additional features until the distance between the new example and all conflicting examples is greater than a parameter c. When c is equal to 0, any difference is sufficient. If c is very large, all features may be measured. The measurement costs of additional features are also taken into account during the discrimination.

5. **Library updating**—Determine whether and how to update an example library given a new example. For example, if a class prediction is incorrect than save the new example.

By instantiating these five functions properly, a variety of CS-learning methods can be designed. These CS-learning methods typically search in two spaces: the traditional concept (description) space common to most learning-from-examples methods, and the example space unique to CS-larning methods. In this latter space, methods explore the descriptive adequacy of sets of features as they seek to minimize the individual expense and overall number of features used. These methods may have different tradeoffs between these two searches, preferring accuracy or efficiency, e.g., by choosing different values d (match distance) and c (discrimination distance). The following two subsections (Tan & Schlimmer, 1990) present two case studies: CS-ID3 (a decision-tree based approach) and CS-IBL (an instance-based approach).

2.1. CS-ID3

CS-ID3 is a CS-learning method based on ID3 (Quinlan, 1986). ID3 is a learning-from-examples method that constructs decision trees to represent concepts. Given a training set of examples with their feature values and classes, ID3 selects a feature to divide the training set according to the possible values of the feature and then recursively builds subtrees to describe partitions. To select features, ID3 applies an information gain measure to estimate the correlation between example features and classes. Specifically, ID3 selects the feature that maximizes information gain I:

$$I(F) = \left[\sum_{i \in \{C\}} - p(C_i) \log_2 p(C_i) \right] - \left[\sum_{i \in F} p(F = V_i) \sum_{j \in \{C\}} -p(C_j | F = V_i) \log_2 p(C_j | F = V_i) \right]$$

where F is a feature, V_i is the ith value of F, $\{C\}$ is the set of classes, and $p(C_j | F = V_i)$ is the proportion of the training examples labeled by C_j and in which F has value V_i. The process terminates when the training examples have the same class or when there are no more features to test. To classify a new object, ID3 repeatedly tests the feature at the current subtree root and follows the matching branch. When it reaches a leaf, it predicts that the new object's class is the most common at the leaf. Although ID3 is a non-incremental method, it can be used incrementally by reprocessing all of the saved examples after every additional example and constructing an entirely new tree.[1]

CS-ID3 modifies ID3 in the following two ways. First, to make ID3 cost-sensitive, CS-ID3's feature selection measure needs to be a function of measurement costs as well as information gain I. CS-ID3 uses the heuristic function $I(F)^2/C(F)$ (Tan & Schlimmer, 1989), where $C(F)$ is the cost of measuring feature F. $I(F)^2/C(F)$ was determined empirically

Table 2. CS-ID3.

Function Name	Definition
Example identification	Identify the examples whose distances are 0 ($d = 0$).
Class prediction	If identified examples have the same class or there are no more features to test, predict the most common class.
Feature selection	Select a feature whose I^2/C is maximal (>0).
Example discrimination	Measure the next cheapest features if there is a class ambiguity at a leaf ($c = 0$).
Library updating	Save the new example.

through several test domains, and its comparison with other cost-sensitive selection functions will be discussed in section 4.

Second, unlike ID3, CS-ID3 only constructs the part of the tree that classifies the current object. This involves measuring only those features of a new object that are determined to be "cost-sensitive" until a unique classification is possible. Note that some of the saved examples may have missing values for features measured in the new object; these partial examples are used to compute the feature selection measure but are otherwise ignored (Tan, 1991b). If there is a prediction error at a leaf, CS-ID3 continues discrimination by measuring the next least expensive feature. If there are any conflicting examples that have the same value as the new object, the ambiguity will remain. In this case, CS-ID3 continues to measure the next least expensive feature, and so on. This lazy evaluation strategy is biased toward using inexpensive features first and then expensive ones if necessary. Because some important features may be missing from saved examples, CS-ID3 may require significantly more examples than ID3 to achieve the same performance. CS-ID3 always saves the new example afterwards. Its definitions for the five functions according to the framework are summarized in table 2.

2.1.1. A simple example

Consider that the goal of a stationary robot is to learn to pick up three types of objects on an assembly line: a cylindrical, closed standing tennis ball can (8 in. tall and 2 in. in diameter), a standing coffee cup (4 in. tall and 2 in. in diameter), and a standing beer cup (6 in. tall and 4 in. in diameter). A sequence of these objects are fed to the robot, one at a time. The sensing procedures that the robot can perform include *Depth* (100 sec. to apply), *Width* (200 sec. to apply), *Height* (300 sec. to apply), and *Front-Hole?* (400 sec. to apply). The grasping classes for the three types of objects are *Ball-Can-Grasp*, *Small-Cup-Grasp*, and *Big-Cup-Grasp*, respectively, and these classes can be determined by some means during incremental learning. CS-ID3 is able to learn a cost-effective decision tree to correctly determine the objects' grasping classes without measuring every possible sensing feature for each object.

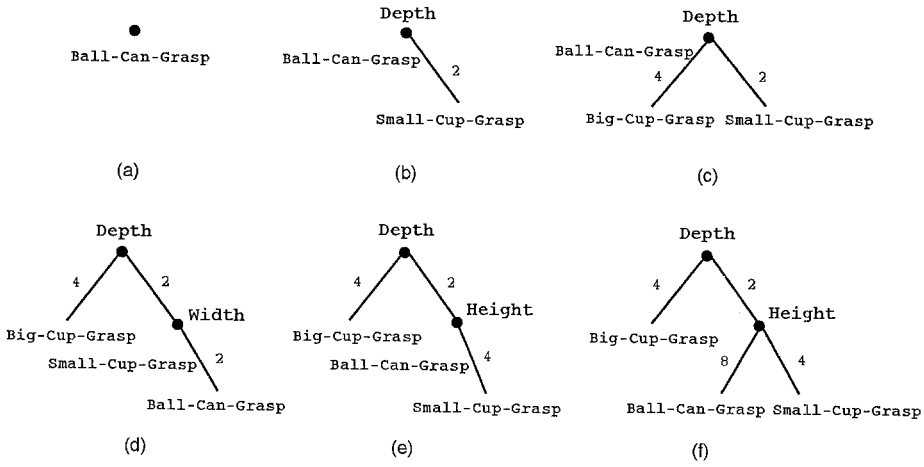

Figure 1. Cost-sensitive decision trees at different stages.

The example library is empty initially. The first object is a standing tennis ball can. Since CS-ID3 has no prediction, it simply saves this empty example and builds a single-node tree (cf. figure 1a). However, the single-node tree fails to classify the second object—a standing coffee cup. The misclassification forces CS-ID3 to measure the cheapest sensing feature *Depth*, which yields a value of 2 inches. The library is augmented by the new example. The tree is expanded to incorporate this new distinction (cf. figure 1b). When the third object, a standing beer cup, is seen, CS-ID3 selects *Depth*, obtains 4 inches, and predicts *Ball-Can-Grasp* (wrong). Since the new example already has a distinct *Depth* value (4), no additional feature is needed for example discrimination. Figure 1c shows the current tree when rebuilt from the three examples. When a tennis ball can is encountered again, CS-ID3 applies *Depth*, obtains 2 inches, and classifies the can as requiring *Small-Cup-Grasp*, which is again incorrect. This misclassification is resolved by introducing the next cheapest sensing feature *Width*. Figure 1d shows the updated tree. The fifth object is a coffee cup. After applying *Depth* and *Width* to the coffee cup, and realizing that it too is unable to discriminate between a tennis ball can and a coffee cup, CS-ID3 tries the next cheapest feature *Height*. In this case, when CS-ID3 rebuilds the tree, it will remove non-discriminatory *Width* (cf. figure 1e). CS-ID3 classifies the next object, a standing beer cup, correctly. After CS-ID3 applies *Height* to the seventh object (a tennis ball can) and classifies it correctly, the tree is converged (cf. figure 1f). After seven objects, CS-ID3 has made five prediction errors (including the no prediction) and saved the partial examples:

{ 1 *Ball-Can-Grasp* [ ] }
{ 2 *Small-Cup-Grasp* [(*Depth*, 2)] }
{ 3 *Big-Cup-Grasp* [(*Depth*, 4)] }
{ 4 *Ball-Can-Grasp* [(*Depth*, 2), (*Width*, 2)] }
{ 5 *Small-Cup-Grasp* [(*Depth*, 2), (*Width*, 2), (*Height*, 4)] }
{ 6 *Big-Cup-Grasp* [(*Depth*, 4)] }
{ 7 *Ball-Can-Grasp* [(*Depth*, 2), (*Height*, 8)] }.

The average cost per object is 229 seconds, and the cost for all objects before convergence is 1600 seconds. Traditional machine learning approaches (e.g., ID3) would measure every feature for each object. Such approaches would make three prediction errors, cost 1000 seconds per object, and need 3000 seconds before convergence. The reason that traditional approaches would take about four times as long per object but only twice as much total time is that they converge quicker than CS-ID3.

2.2. CS-IBL

Like ID3, the instance-based-learning (IBL) methods learn from examples (Aha, 1989). Instead of constructing an explicit, abstract representation of the training examples, IBL methods simply save examples and rely on abstract matching. Given a new object to classify, they find the most similar example in memory and predict its class for the new object. For the sake of convenience, the specific IBL method studied here is referred as IBL. The similarity measure it uses is the negation of the Euclidean distance between two examples (i.e., the same definition described under the generic function Example Identification in section 2). It saves only examples that were incorrectly predicted.

To make IBL cost-sensitive, the classification process it uses to find similar examples must specify which features (and how many) should be measured for a new object. Following the spirit of IBL, CS-IBL uses saved examples to serve as templates for feature evaluation. Instead of measuring all features of all saved examples, CS-IBL repeatedly selects one of the saved examples and one of the features with a known value for that example and measures that feature for the new object until the closest example has been found. There are three steps in each selection cycle.

First, CS-IBL identifies the saved examples that are similar to a new object. Specifically, they are the examples whose distances differ from the distance of the closest saved example by no more than d. In the beginning, all saved examples have CS-IBL's attention.

Second, CS-IBL chooses one of the identified examples that has features that are not yet measured for the new object, have common values, and are inexpensive. Specifically, CS-IBL chooses the example that maximizes the ratio of expected match success to expected match cost. The expected match success is estimated by the product of independent feature-value probabilities. The expected match cost of an identified example is defined as the expected cost of measuring its features, according to the decreasing order defined below, until all the features have been found to match the values of a new example or until a mismatch is found. Equation 1 implements this ratio for the saved example i, where P_{ij} is the estimated likelihood that feature F_j will have the same value as the one in the example i based on all saved examples, $\{F'\}$ is the set of features measured for the saved example but not for the new object and sorted in decreasing order of $P_{ij}/C(F_j)$ (cf. Cox, 1989), and $C(F_j)$ is the cost of measuring feature F_j:

$$\frac{\Pi_{j\in\{F'\}}\ P_{ij}}{\Sigma_{j\in\{F'\}}\ C(F_j)\ \times\ \Pi_{k=1}^{j-1}\ P_{ik}} \tag{1}$$

To favor the example that has the minimal distance among identified examples, equation 1 is multiplied by a discount factor:

$$1 - \frac{example\_distance - min\_distance}{max\_distance - min\_distance + 1} \quad (2)$$

where *example_distance* is the distance between the identified example and the new example, and *min_distance* (*max_distance*) is the minimal (maximal) distance among the identified examples. The "+1" ensures that the denominator will not become zero.

Third, given the chosen example to guide feature selection, CS-IBL measures one of its features that has a likely value and is inexpensive, or maximizes $P_{ij}/C(F_j)$. The intuitive motivation behind these selection heuristics is that a matching algorithm should first try to match a new object to "close" and "easily-matched" examples starting with "easy" features. An alternative strategy is to start with "hard" features that have unlikely values and are inexpensive, to eliminate the chosen example early when the "hard" features are mismatched.

CS-IBL repeats these three steps until the upper bound on the distance from the chosen example is less than the lower bound for one from any other class. This stopping criteria is conservative and reminiscent of Gennari's (1989) criteria for focus-of-attention in COBWEB (Fisher, 1987). A final modification ensures that CS-IBL does not end up storing a set of empty or insufficiently described examples. After a prediction error, CS-IBL measures one or more additional features for the new object such that the distance between the new example and any conflicting example is greater than c. Additional features to measure are first drawn from the closest conflicting example if any and then form new features (difference 0.5), on a cost basis. CS-IBL's defintions for the five functions according to the framework are summarized in table 3.

For the simple example given in section 2.1.1, CS-IBL measures sensing features in a manner similar to CS-ID3, except that CS-IBL saves fewer examples. Additional differences between them will be discussed in the following subsection and also in section 3.2.

Table 3. CS-IBL.

Function Name	Definition
Example identification	Identify the examples whose distances differ from the distance of the closest example by no more than d.
Class prediction	If the upper bound on the distance from a chosen example is less than the lower bound for one from any other class, predict the class of the chosen example.
Feature selection	Select a feature that has a likely value and is inexpensive from the identified example that maximizes the ratio of expected match success to cost.
Example discrimination	Measure the next cheapest features such that the distance between the new example and any conflicting example is greater than c.
Library updating	If a prediction is incorrect, save the new example.

2.3 Comparative experiments in a synthetic domain

One might expect that incremental learning-from-examples methods that measure all example features are more expensive and incur fewer errors prior to convergence than CS-learning methods. The empirical results of this subsection bear this out. We ask two open questions: how efficient are the two methods, and how many errors do they incur prior to convergence given different feature cost distributions and numbers of irrelevant features? To provide initial answers to these questions, ID3, CS-ID3, IBL, and CS-IBL are compared in a synthetic domain. For these studies, ID3 is applied as a brute-force incremental method, rebuilding the tree from scratch after predicting the class of each new object. Both ID3 and IBL can be implemented as instances of the framework. ID3's implementation uses an extreme discrimination function ($c = \infty$) to measure all remaining features, and IBL's implementation uses a void class prediction function to delay a prediction until all example features have been measured. For comparison, CS-IBL has its parameters, d and c, set to 0. It is the most cost-sensitive version.

The experiments use a simple Boolean concept, $(A \wedge B) \vee (C \wedge D)$. In this study, the values of (irrelevant) features are randomly chosen. Given four irrelevant features and moderately uneven costs (condition 3 below), figure 2 depicts total cumulative errors prior to convergence for each of the four methods ID3, CS-ID3, IBL, and CS-IBL. Results are averaged over five random executions; bars denote standard deviation. ID3 yields the fewest errors; IBL comes in second (also the slowest to converge), with CS-ID3 and CS-IBL following behind.

For the same conditions, figure 3 depicts the number of features measured per example by each of the methods as training progresses. Note that at 1500 examples, IBL still had not converged (see figure 2). The CS-learning methods measure considerably fewer features than ID3 and IBL. As both CS-learning methods search the space of examples, CS-ID3 settles to measuring a near minimum number of features (calculated by hand), but CS-IBL does not. I suspect that this is an artifact of CS-IBL's conservative stopping criteria.

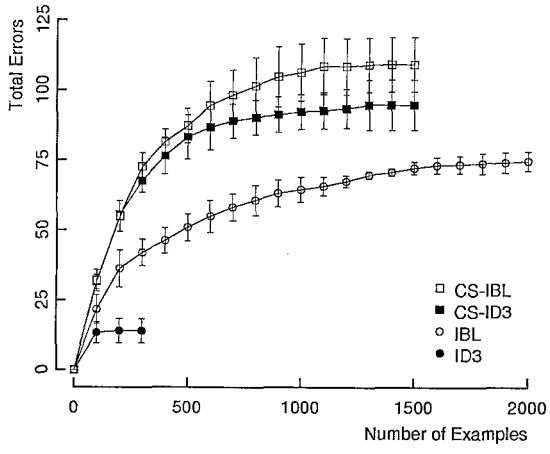

Figure 2. Total errors prior to convergence given four irrelevant features and moderately uneven feature costs.

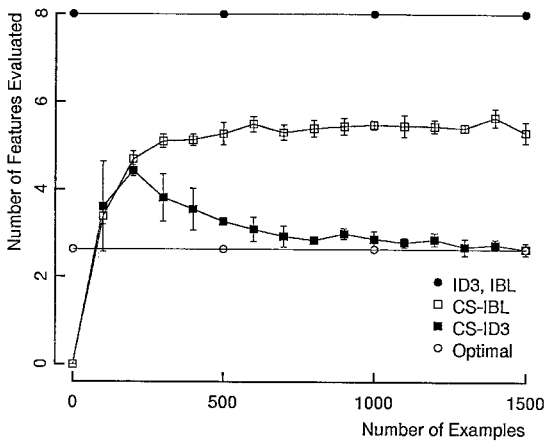

Figure 3. Average number of features measured per example prior to convergence given four irrelevant features and moderately uneven feature costs.

CS-learning methods should be sensitive to differential feature costs. Specifically, they should use less expensive features when they are effective. We can verify this by varying the relative costs of features and then observing the resulting behavior of the four methods. Standard deviation is one quantitative measure of feature cost skew; identical costs measure zero, and uneven costs measure much greater than one (when some features are much less expensive than others). Here, standard deviation (σ) is used to measure feature cost distributions. Using the simple Boolean concept above, 40 cost units are randomly divided among groups of four features to yield four conditions: (1) each feature costs 10 (σ = 0), (2) 1 feature costs 1, and 3 features cost 13 (σ = 5.6), (3) 2 features cost 1, and 2 features cost 19 (σ = 9.6),[2] and (4) 3 features cost 1, and 1 feature costs 37 (σ = 16.7). When irelevant features are included, they are assigned the same costs as relevant features. Given four irrelevant features, figure 4 depicts the average cost per example prior to convergence, and figure 5 shows total errors prior to convergence as a function of different relative feature costs for each of the methods. In terms of average cost, both CS-learning methods exhibit close to optimal asymptotic performance. They are also considerably less than ID3 and IBL. In terms of errors, the methods separate naturally into three groups, from best to worst: ID3, IBL, and CS-learning methods. The poor performance of the CS-learning methods may arise because they must search for an effective set of features to measure.

CS-learning methods should also be sensitive to the number of irrelevant features, as focus-of-attention methods are (Gennari, 1989). Using the simple Boolean concept, 0, 2, 4, and 8 irrelevant features were added that have random binary values. Given moderately uneven costs (condition 3), figure 6 depicts average cost per example prior to convergence, and figure 7 shows total errors prior to convergence as a function of the number of irrelevant features. In terms of average costs, the CS-learning methods again perform close to the optimal *asymptotic* level.[3] In terms of errors, the methods appear to fall into three groups, from best to worst: ID3, CS-ID3, and the instance-based methods. Both IBL and CS-IBL incur a sharply increasing number of errors as irrelevant features increase, something that

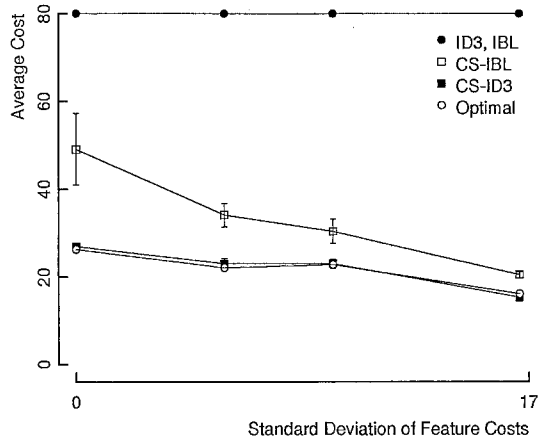

Figure 4. Average cost per example prior to convergence given four irrelevant features.

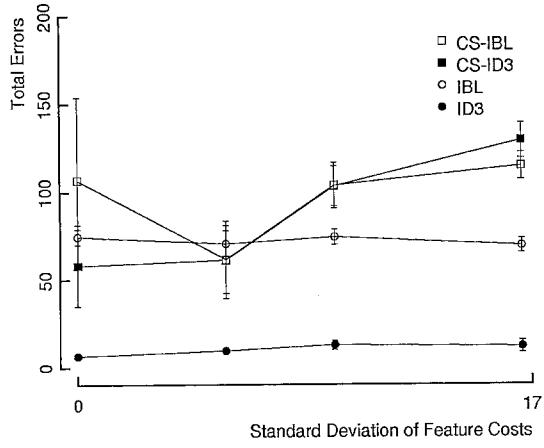

Figure 5. Total errors prior to convergence given four irrelevant features.

may be remedied by more sophisticated version of IBL (Aha & Goldstone, 1990). Unlike the fact that ID3 has better performance than CS-ID3, IBL and CS-IBL appear to have no significant differences.

In this synthetic domain, CS-learning methods were shown to be considerably more efficient per example and to sacrifice little asymptotic accuracy, but to incur more errors prior to convergence than prior methods. The experiments also imply that CS-learning methods are more efficient than prior methods that are insensitive to feature costs after convergence. If asymptotic performance were the only concern, using ID3 (IBL) for learning and then using CS-ID3 (CS-IBL) for classification after convergence would be equally effective. However, if prediction errors can be promptly detected and corrected efficiently during

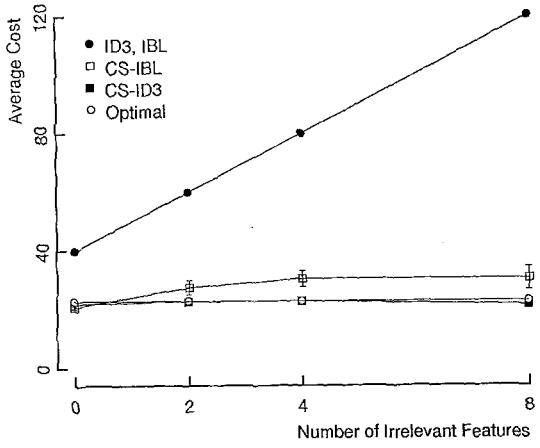

Figure 6. Average cost per example prior to convergence given moderately uneven costs.

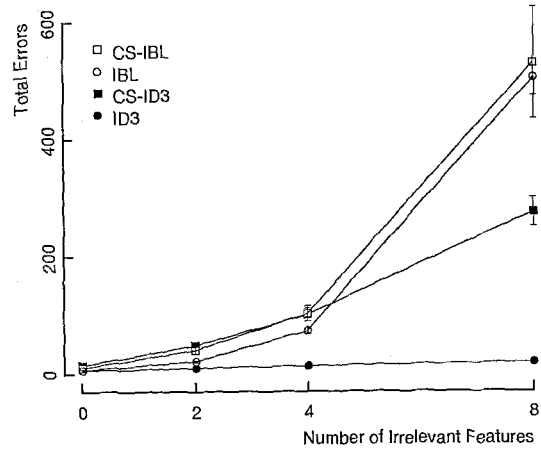

Figure 7. Total errors prior to convergence given moderately uneven costs.

learning, then CS-learning methods can be used effectively for robotic classification tasks, or any other similar tasks, where exhaustive sensing (or measurement) is too expensive during incremental learning or there is a cost (time) limit to perform on each object.

3. Applying cost-sensitive learning to an approach-recognize task

The study of knowledge-poor sensing and grasping strategies is viable for such diverse domains as industrial assembly, undersea salvage, nuclear waste cleanup, and planetary exploration. The following approach–recognize task is simplified from such tasks and is used as the realistic testbed for CS-learning.

Given a set of unknown objects on the floor of a research laboratory, the primary task of a mobile robot is to pick up the objects and discard them in a waste box quickly (see figure 8). The robot agent is a modified Health/Zenith Hero 2000, a mobile robot with a five-degree of freedom arm and a two-finger gripper. The Hero robot has a wrist-mounted sonar that can be repositioned relative to the robot body and a touch sensor able to detect the failure of a grasping action. The Hero robot has access to a ceiling-mounted camera that is able to view the entire laboratory through a fish-eye lens. Since the resolution of an overhead image is fairly coarse and the image is 2-D, the vision subsystem can locate object candidates and plan approaching paths but not identify them. The robot has to navigate close to each of these object candidates and use its wrist-mounted sonar to identify them in turn.

At the robot's disposal are a set of fixed sonar-based sensing procedures (see figure 9 and table 4), a set of moving procedures, and a library of fixed grasping procedures (see figure 10 and table 5). Each sensing or grasping procedure requires two parameters: distance

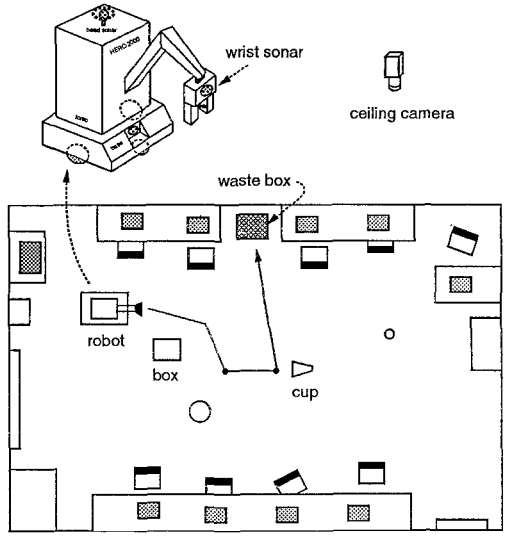

Figure 8. Mobile robot laboratory setup.

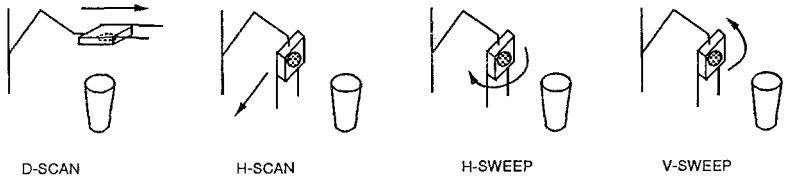

Figure 9. Sensing procedure types.

Table 4. Basic knowledge of the sensing procedures.

Procedure	Range	Speed	Error	Features
H-Sweep-1°	6–127 in.	100 sec.	±3°	*Width-Degree-1°*
H-Sweep-5°	6–127 in.	45 sec.	±5°	*Width-Degree-5°*
V-Sweep-1°	6–127 in.	76 sec.	±3°	*Height-Degree-1°*
V-Sweep-5°	6–127 in.	42 sec.	±5°	*Height-Degree-5°*
S-Sweep-1°	6 in.	70 sec.	±3°	*Depth-Degree-1°*
H-Scan-0.3	6 in.	105 sec.	±0.5 in.	*Width, Front-Hole?*
D-Scan-0.3	6 in.	84 sec.	±0.5 in.	*Height, Depth, Top-Hole?*

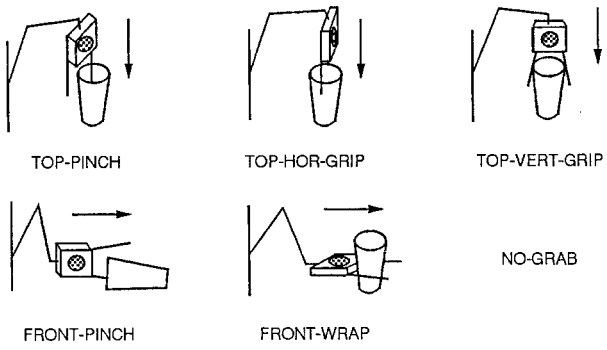

Figure 10. Grasping procedures.

Table 5. Basic knowledge of the grasping procedures.

Procedure	Range	Speed
Front-Pinch	6 in.	45 sec.
Front-Wrap	6 in.	60 sec.
Top-Pinch	6 in.	63 sec.
Top-Vert-Grip	6 in.	80 sec.
Top-Hor-Grip	6 in.	80 sec.
No-Grab	6–999 in.	0 sec.

(within its range) and orientation. The expected error of each sensing procedure is given, and the costs of sensing, moving, and grasping procedures are assumed known. For example, depending on a moving distance, the amount of vision processing, the number of sonar sweeps involved for precise positioning, moving can take up to several minutes. Some sensing procedures produce (e.g., *D-Scan-0.3*) more than one feature for a fixed cost. If one of such features is selected and measured, it also yields additional information for the rest of the features given by the same sensing procedure without additional expense. The robot must be able to distinguish graspable objects from ungraspable objects, decide which grasping procedure to use if required, and be flexible enough to pick up objects anywhere in the open space. In addition, it must minimize the execution cost.

Since depositing a grasped object into a known waste box is relatively straightforward, the main task for the robot is to find a classification procedure that provides the mapping from sensing procedures to grasping procedures. If one had complete knowledge of objects and could anticipate possible variation in an environment, then an efficient classification procedure could be defined by hand, but programming a flexible classification procedure describing such a mapping can be difficult due to the variations of objects, their positions, their surrounding conditions (e.g., obstacles), and the robot itself. For instance, moving costs may change depending on path conditions. In addition, if novel objects may be encountered, approaches based on known object models may be insufficient. This justifies a learning approach.

Traditional learning-from-examples methods would be a natural candidate, where object descriptions map to examples, sensing procedures map to features, and appropriate grasping procedures map to classes. Such methods require little domain knowledge and can handle sensing noise (figure 11 shows a typical noisy sonar map for a cup). However, learning the necessary mapping from sensing procedures to appropriate grasping procedures in this domain is beyond current learning-from-examples methods due to the following factors:

- Object descriptions are potentially quite large. Each object may be described by the results of the many parameter instantiations of all possible sensing procedures. Collecting exhaustive object descriptions can be unreasonably expensive, or even impossible.
- Object descriptions are initially empty. The robot must decide which sensing procedures to execute and where for any given object.
- Because sensing procedures have significant sensing costs (see table 4), the ability of a particular sensor feature to discriminate between appropriate grasping procedures must be balanced against the execution cost of its corresponding sensing procedure.

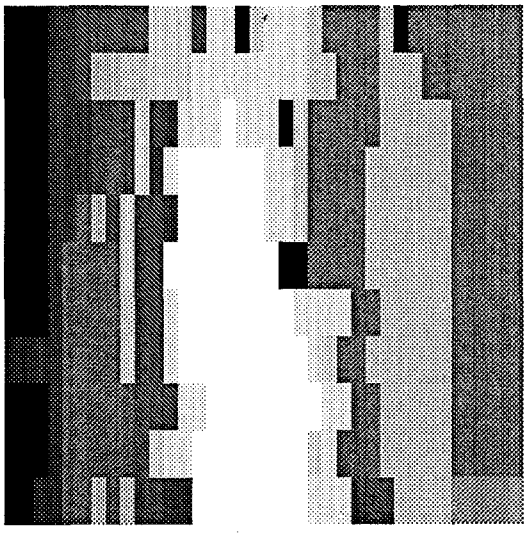

Figure 11. A sonar map of a standing small cup.

For these reasons, CS-ID3 (cf. section 2.1) is used to control the Hero robot for the approach–recognize task. The resulting CS-learning robot, CS-HERO, collects object descriptions from its own sensing and experimentation, and performs active perception using a learned cost-sensitive decision tree. Furthermore, CS-HERO is able to tolerate sensing noise, and is able to detect grasping failures and recover from them.

Because objects' appearances vary with viewing angle and distance, sensing procedures yield different values at different positions. In this implementation, each specific feature is associated with a position described by two parameters: a distance and an orientation between the wrist-sonar sensor and an object. A distance can be measured by the wrist sonar. The orientation of an object (or the robot itself) is determined by the major axis of the object's 2-D image. Since the resulting major axis has two directions, the orientation is defined as the one associated with the half having a larger number of pixels. Potential positions for sensing features are determined by object-sensitive heuristics, which are not presented here (Tan & Schlimmer, 1991). The run-time cost of a position-dependent feature is its cost plus the cost in moving from the current robot position to the sensing position. Similarly, grasping classes are also associated with chosen grasping positions.

Table 6 shows the simplified version of the algorithm of CS-HERO, already an extension of CS-ID3. First, the vision subsystem is utilized to locate the Hero robot and a new object. Then CS-HERO executes a moving procedure, possibly involving path planning, to approach the new object. While approaching, the robot occasionally stops and applies sensing procedures, selected by I^2/C, at their sensing positions to obtain the values of the corresponding features (step 3(a, b, c)). Like CS-ID3, the robot repeatedly uses the obtained values to partition the saved object descriptions/examples (step 3(d)) until it can predict a (position-dependent) grasping class (procedure) (step 3(e)). If CS-HERO cannot predict a class within the expected errors of the measured features (step 3(f)), it reselects features and follows the nearest branch at each node to predict a class. If a predicted grasping procedure fails to pick up the new object (assuming no side effect on the object after grasping; see Tan (1991b) for how to handle side effects), CS-HERO rebuilds a decision tree from the object descriptions that do not have the failed grasping class (step 4). When the remaining object descriptions become null, it simply guesses a new grasping class that has not been applied. If the new object description cannot be distinguished from a conflicting saved object description, CS-HERO measures additional sensing features on a cost base and records their values in the new object description (step 5). Finally, it saves the new object description in its library (step 6) (trees are not saved, and they are generated on the fly each time).

Figure 12 shows one of its decision trees after CS-HERO has been trained on six objects. Internal nodes in the tree are position dependent sensing features, and leaves correspond to position dependent grasping classes. Due to moving cost, CS-HERO selects different sensing features at different (initial) positions. Each branch in the tree represents a possible motion command, and its moving cost in seconds is given in a parenthesis. Each branch is also labeled by a value range (e.g., [7, 8]) after CS-HERO merges any two values of a feature, whose difference is within its expected error, into the same branch. Suppose that the robot starts at the initial position of distance 35 inches and orientation 250 degrees from an object, CS-HERO selects *Height-Degree-I°* (26″, 270°). The robot takes about 53 seconds to move to the sensing position (26″, 270°) and 76 seconds to acquire the *Height-Degree-I°* value. Assuming the outcome is 17°, CS-HERO follows the matching branch,

Table 6. The algorithm of the learning robot CS-HERO.

For each new object:

1. Let CURRENT-ODS be the object descriptions saved in a library.
2. Let IDENTIFIED-ODS be CURRENT-ODS.
3. Repeat
   (a) Determine relative distance and orientation to the object.
   (b) Select a sensing feature from IDENTIFIED-ODS according to
   $$\frac{information\_gain^2}{sensing\_cost + moving\_cost}.$$
   (c) Move the robot to the sensing position and record the feature's value in the new object description.
   (d) Remove the object descriptions that do not match the feature's value (beyond its expected error) from IDENTIFIED-ODS.

   until
   (e) There is only one remaining grasping class or there are no features left in IDENTIFIED-ODS, predict the most common class and go to 4, or
   (f) IDENTIFIED-ODS is null, replace the 'expected error' match by the nearest neighbor match, and go to 2. (The features with known values will be selected first because they have zero cost now.)
4. If the predicted grasping class is executed successfully, add this class name into the new object description and go to 5. Otherwise, remove the object descriptions having the failed grasping class from CURRENT-ODS. If CURRENT-ODS becomes null, predict a new grasping class and go to 4. Otherwise, go to 2.
5. If the feature values of the new object description are included in the feature values of a saved object description that has a different grasping class, apply the next cheapest sensing procedure and record the corresponding feature value in the new object description, go to 5. Otherwise, go to 6.
6. Save the new object description in the library.

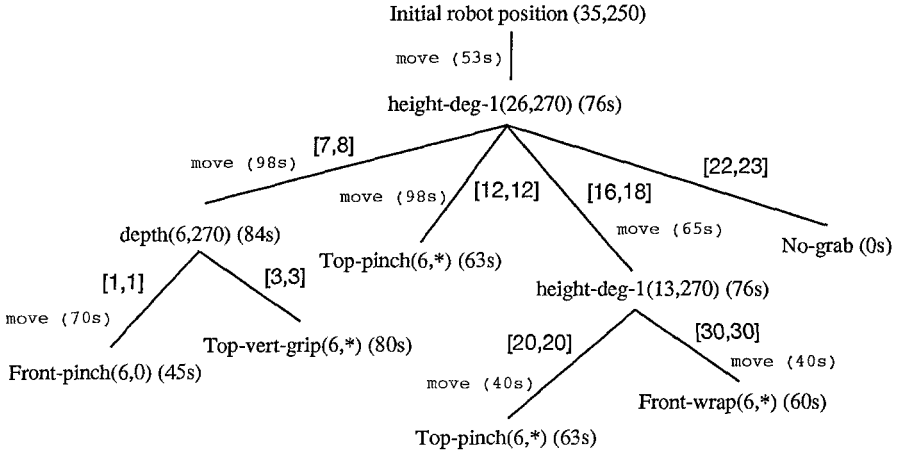

Figure 12. A CS-HERO's decision tree for distance = 35″ and orientation = 250°.

and the robot moves to (13″, 270°) to measure *Height-Degree-1*° (13″, 270°). If its value is 30°, then CS-HERO predicts *Front-Wrap* (6″, *) and takes 100 seconds to move close to and grasp the object by *Front-Wrap*. The symbol * means "don't care" (for orientation here).

3.1. Robot experiments

Several experiments were conducted with CS-HERO. The objects used in the experiments included cups of different sizes, open or closed cardboard boxes of different sizes, tennis ball cans, soda cans, magazine boxes, and bottles. Each experimental object could be treated as several different objects depending on its placement. For example, a cup could be standing upright, on its side, or upside down. An open box could have its opening facing any of six directions.

CS-HERO was given over two dozen objects (see table 7). Among them, 14 objects were presented to CS-HERO repeatedly for training and another 11 for testing. Object-sensitive

Table 7. Set of objects for the expanded experiment.

	Grasping Procedure(s)
Training objects	
8 oz. Styrofoam coffee cup	*Front-Wrap*
12 oz. soda can (upright)	*Front-Wrap*
12 oz. soda can (lying down)	*Top-Vert-Grip*, *Top-Hor-Grip*
16 oz. short plastic tumbler (upright)	*Top-Pinch*
16 oz. short plastic tumbler (lying down)	*Front-Pinch*
32 oz. plastic tumbler (upright)	*Top-Pinch*
32 oz. plastic tumbler (lying down)	*Front-Pinch*
Tennis ball can (upright)	*Front-Warp*
Tennis ball can (lying down)	*Top-Vert-Grip*, *Top-Hor-Grip*
9×4×5 in. open cardboard box	
(upright, facing sideways)	*Front-Pinch*
(lying down, facing up)	*Top-Pinch*
(lying down, facing sideways)	*Front-Pinch*
(lying down, facing down)	*No-Grab*
11×9×12 in. closed cardboard box	*No-Grab*
Test objects	
2×2×2 in. closed cardboard box	*Front-Wrap*
16 oz. soda bottle (upright)	*Front-Wrap*
16 oz. soda bottle (lying down)	*Top-Vert-Grip*, *Top-Hor-Grip*
20 oz. plastic tumbler (upright)	*Top-Pinch*
20 oz. plastic tumbler (lying down)	*Front-Pinch*
20 oz. plastic tumbler (upside down)	*Top-Vert-Grip*
9×4×7 in. cardboard magazine box	
(upright, facing sideways)	*Front-Pinch*
(upright, facing up)	*Top-Pinch*
(lying down, facing sideways)	*Front-Pinch*
(upside down, facing sideways)	*Front-Pinch*
10×10×10 in. closed cardboard box	*No-Grab*

heuristics (Tan & Schlimmer, 1991) determined that CS-HERO should use four sensing orientations 0°, 90°, 180°, and 270°, and for each of the sensing orientations, use three sensing distances 6, 16, and 24 inches. Therefore, there were 72 ($12 \times 4 + 4 \times 6$) available position-dependent sensing features (see table 4). Some objects have two grasping classes depending on the robot's grasping position (e.g., *Top-Hor-Grip* for (6", 0°) and *Top-Vert-Grip* for (6", 90°)).

After training, CS-HERO was able to grasp most of the training objects successfully. On one occasion, the *Width*(6", 90°) value of a lying-down soda can was measured incorrectly, resulting in an inappropriate grasping procedure *Front-Wrap*(6", 90°). It failed because a lying-down soda can was too wide for the gripper to grasp it from its long side (5 in.). A lying-down soda can rolled away as the result of applying *Front-Wrap*(6", 90°). CS-HERO was able to track down the soda can using vision, rebuild a decision path from a current position, and pick it up by executing the next most likely grasping procedure—either *Top-Vert-Grip* or *Top-Hor-Grip*, depending on the robot's approaching direction.

To find out the generalization ability of CS-HERO's decision trees, CS-HERO's performance was tested on unseen test objects. It succeeded in grasping 9 out of 11 on the first attempt. The overall success rate of CS-HERO was over 80%. Many grasping failures can be prevented in the future through incremental learning. For example, one failure occurred for the upside-down 20-oz. plastic tumbler. During the classification, CS-HERO only measured the shape of a cup-like object and determined the upside-down cup was similar enough to an upright cup. As a result, CS-HERO output grasping class *Top-Pinch*(6", *) and only found Hero's fingers jammed by the bottom of the cup. Eventually, by failure recover (step 4 in table 6), it found out that the correct grasping class was *Top-Vert-Grip*(6", *). To prevent the same failure in the future, CS-HERO disambiguated the upside-down cup from other upright cups by measuring additional features. In this case, CS-HERO realized that the upside-down cup had no hole on the top and recorded its *Top-Hole?* value in that position in the new object description. The next time around, CS-HERO would use this specific feature *Top-Hole?* to distinguish upside-down cups from upright cups. Indeed, it succeeded in the next run.

In addition to being able to accomplish the object grasping task, the classification procedures formed by CS-HERO also appear to be highly efficient, averaging 12 minutes per object (including 20 seconds computation time). Sensing features were selected according to their relevance to the classification. For example, for an open box facing sideways, sensing from one orientation (viewing angle) is often inadequate to find its opening. CS-HERO learned to sense from different orientations to determine whether an unknown box was open or closed. CS-HERO often used only a small number of sensing features. If CS-HERO measured every position-dependent feature in existing object descriptions, as ID3 would do for brute-force incremental learning, it would take over 90 minutes to classify each object. Even given the same object descriptions, the average cost decision trees (using I^2/C) were at least 30% faster (a reduction of 30%) than standard decision trees (using I only) in terms of average execution cost after convergence.

Clearly, in this domain, CS-HERO is more efficient than incremental learning-from-examples methods that measure all example features, but it may incur more errors prior to convergence (i.e., prior to correctly classifying all encountered objects) than the standard methods. Although the robot experiments are realistic, it is difficult to accurately assess

this tradeoff in the robot domain because there are so many test conditions, sensing noise occurs irregularly, and the cost of recovering from a grasping failure varies. In contrast, simulations can afford precise control over experimental conditions and can provide this insightful information approximately.

3.2. Robot simulations

In the following simulations, complete object data were first collected from the 16 training objects (including the 14 training objects in table 7 plus the 9×4×5 in. open cardboard box with an opening facing two additional directions) at each of the 12 sensing positions. Given initial robot positions of distance 10 or 35 inches and four preferred orientations, performance data were measured as an average over eight random object orders. I assumed no sensing noise, and the costs of grasping procedures were ignored. For comparison brute-force incremental ID3, CS-ID3 (used by CS-HERO), IBL, and CS-IBL were applied to the approach-recognize task. Table 8 summarizes five dependent measures prior to convergence: (a) the average measurement cost per object, (b) the total errors incurred, (c) the number of examples required, (d) the size of the example library, and (e) the total cost.

CS-ID3 and CS-IBL are highly cost-efficient representing an order of magnitude savings compared to ID3's and IBL's strategy of measuring all features for each example. In terms of errors ID3 and IBL outperform their cost-sensitive derivatives approximately three to one. About half the errors made by CS-ID3 and CS-IBL were attributed to two pairs of objects derived from the same open box with an opening facing opposite directions (0° and 180°, 90° and 270°). Each pair had the same values for most features. CS-ID3 and CS-IBL made half of their errors when searching for the discriminative features (e.g., *Front-Hole?*(6″, 0°)). Nevertheless, if a correct grasping procedure after a prediction error can always be determined and executed within 75 minutes (i.e., (5030 − 493)/60 ≈ 75 from table 8), then CS-ID3 (CS-IBL) is superior to ID3 (IBL) overall. This is true for the robot domain. Each assuming that a recovery from a prediction error is very expensive, the cost benefit by CS-ID3 (CS-IBL) will pay off gradually after convergence (since CS-ID3 (CS-IBL) saves 4566 (4537) seconds per object without making any error). Note that the four total costs prior to convergence are comparable in this domain. In general, there is no clear preference between CS-learning methods and standard methods in terms of the total cost prior to convergence. This measurement depends on specific domain (e.g., number of irrelevant features) and method (e.g., search biases).

Table 8. Performance of the learning methods in the robot domain.

Prior to Convergence	ID3	CS-ID3	IBL	CS-IBL
Average cost (seconds)	5030	464	5030	493
Total errors	12	36	12	40
Number of examples	16	204	18	190
Library size	16	204	13	41
Total cost	80480	94656	90540	93670

CS-ID3 and CS-IBL have comparable performance in terms of costs and errors in the robot domain. One primary difference is that CS-ID3 saves all examples while CS-IBL does not save those whose classes were correctly predicted. In the experiment, the library size of CS-ID3 was five times larger than that of CS-IBL. On the other hand, if CS-IBL is misled by the cost-driven heuristics to measure an irrelevant feature value, the mistake is subsequently propagated, and whenever a saved example with that feature is selected, the irrelevant feature will be remeasured for a new example. Though not as bad as measuring all irrelevant features (as IBL does), it is not as good as measuring none (as CS-ID3 does). This characteristic probably accounted for the slight cost increase in CS-IBL. At an abstract level, one difference between the two methods is that CS-IBL avoids committing to a particular test ordering, and I suspect that this may make it easier to tolerate feature noise and dynamic changes in feature costs.

4. Related work

The importance of execution costs has been observed by several researchers in robotics. For model-based object classification, Hong, Ikeuchi, and Gremban (1990) have designed an optimizing vision algorithm compiler that determines the minimum-cost sequence of operations needed to classify an object. Goldberg and Mason (1990) have applied a similar technique to actions in a frictionless parallel-jaw gripper domain. Chen and Mulgaonkar (1990) have proposed a feature-utility measure for automatic vision programming; their algorithm selects seed features based on the ratios of matching costs to reliabilities. Dean et al. (1990) and Miller (1989) have studied selecting sensing operations by using techniques similar to those for selecting actions in robot planning and control. Their selection criteria are based on known probability distributions and execution costs. However, none of these systems address the issue of learning classification knowledge that automatically generates descriptions that characterize a given collection of objects or a subset of them.

Before discussing the related work in machine learning, let me outline four spaces in concept learning: 1) feature space, in which each feature (or a constructed feature) describes a property of an object (e.g., the *Height* of an object), 2) example space, in which each (incomplete) example consists of a subset of features (and their values) describing an object, 3) concept space (extensional space), in which each concept covers a specific subset of examples, and 4) concept description space (intensional space), in which each concept description is a logical combination of features defining a concept. Figure 13 illustrates these four spaces. As illustrated, example *e3* consists of features *f1*, *f2*, and *f3*; concept *C* covers examples *e1*, *e2*, and *e3*; each of the descriptions *d1*, *d2*, and *d3* defines the same concept *C*.

Traditional learning-from-examples methods (Breiman et al., 1984; Feigenbaum & Simon, 1984; Michalski & Chilausky, 1980; Mitchell, 1977; Quinlan, 1986) search through the concept space (Mitchell, 1982) (see the lightly shaded connection in figure 13). They aim to improve the accuracy of a learned concept. Although some of them do bias towards short concept descriptions, these biases do not change according to input. Therefore, they effectively equate a concept with its description. In other words, the relationship between a concept space and a concept description space in these methods remains one-to-one. As

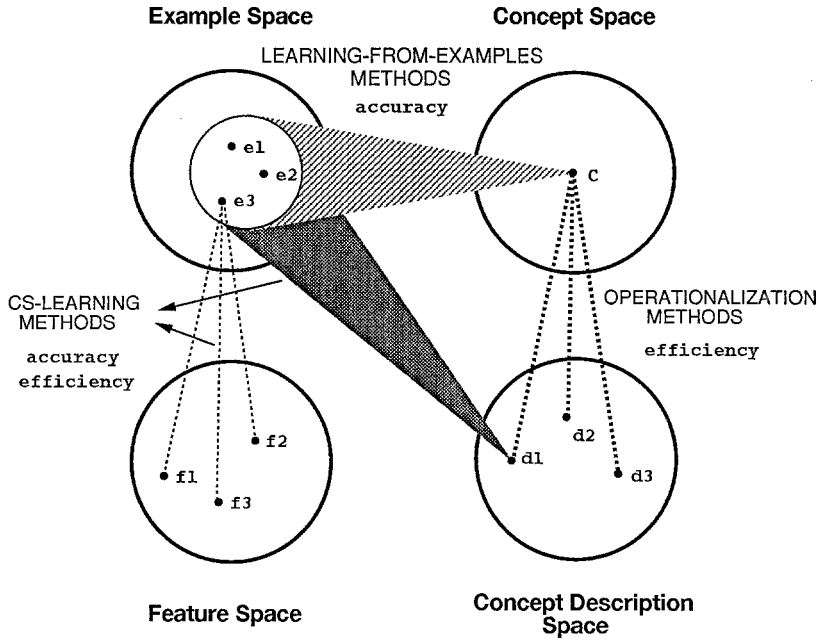

Figure 13. Four different spaces in concept learning.

a result, their learned concept descriptions may not be efficient ones. On the other hand, operationalization methods like EBG (Laird, Rosenbloom, & Newell, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986) can be viewed as a search through a concept description space (Keller, 1987) (see the thick, dotted lines in figure 13). Given a non-operational description, an operationalization method re-expresses it by an operational (efficient) description through explanation using domain knowledge. Keller (1987) defined the operationality for explanation-based learning in terms of *usability* and *utility*. He suggested that an explanation should be constructed only to the level that the primitives of a concept description could be used efficiently by the performance system. The efficiency, not accuracy, of a concept description is the major issue for operationalization methods.

Unlike either traditional learning-from-examples or operationalization methods, CS-learning methods (e.g., CS-ID3) consider both accuracy and efficiency during learning of a concept description. In fact, they search through both a concept description space and an example space. Given a set of examples and feature measurement costs, CS-learning methods search through a concept description space for an efficient description (see the darkly shaded connection in figure 13). One difference between CS-learning methods and operationalization methods in finding an efficient description is that the former rely on feature selection (e.g., *Height* vs. *Width*) and the later use feature operationality (e.g., *Stable* vs. *A-flat-bottom*). Nunez (1991) and Norton (1989) also extended the selection function of ID3 to $(2^I - 1)/(C + 1)$ and I/C, respectively, where I is the information gain and C is the measurement cost of a feature. Although their heuristic functions bear some difference to I^2/C used in CS-ID3, pilot studies indicate that using any of these three functions

makes no difference in terms of (non-incremental) performance in our domains (Tan, 1992b). The main difference between CS-ID3 and theirs is that their methods try to build an efficient tree from a batch of (presumably) complete examples while CS-ID3 tries to do the same incrementally with partially constructed examples. The major contribution of CS-learning methods amounts to considering costs in an incremental context during learning. In other words, CS-learning methods perform an additional search in an example space (see the thin, dotted lines in figure 13). Presumably, the goal state of such a search is a constructed example that consists of discriminative and cheap features, and operators select features. The justification for this search is that including too many features in examples can be too expensive for example construction. On the other hand, examples with too few features can be insufficient for subsequent learning.

5. Limitations and conclusions

This article investigates the impact of realistic costs on learning-from-examples methods and addresses the issue of how to reduce the overall cost of measuring features during both learning and use of classification knowledge. It presents a unified framework to characterize a class of cost-sensitive-learning methods (i.e., CS-learning methods) that can trade off accuracy for efficiency during learning, and empirically demonstrates such tradeoffs for decision-tree-based (CS-ID3) and instance-based (CS-IBL) CS-learning methods.

However, this study of CS-learning methods is just a beginning. We have only explored one synthetic domain and one robot domain. We do not know how concept complexity relates to the cost savings of CS-learning methods. No experiments were done to determine how the settings of the parameters (d and c) affect such savings quantitatively, not to mention how to select them. For example, if there is a significant recovery cost associated with prediction errors during learning (e.g., in some medical diagnosis domains), CS-ID3 using complete examples (e.g., by setting parameter $c = \infty$) can outperform CS-ID3 using partially constructed examples (e.g., $c = 0$) because the latter makes more errors than the former prior to convergence. Furthermore, CS-learning methods only accept point cost values and describe an object by a set of feature-value pairs without structure. CS-learning methods' total costs prior to convergence may or may not be less than the ones of their non-cost-sensitive counterparts. The tradeoff between the total cost prior to convergence and the asymptotic cost per object (after convergence) has not been studied. Finally, we need an analytical approach to determine the gain of CS-learning method in various domains.

Nevertheless, CS-learning methods are 1) *incremental* (they also can adapt to newly introduced features), 2) *cost-sensitive* before and after convergence, 3) *simple* (modifying some existing learning-from-examples methods), and 4) *flexible* (being able to trade off between the number of cumulative prediction errors (accuracy) and the average measurement cost per object (efficiency) during learning). Moreover, CS-learning methods can actively seek information about objects, and change their focus of attention as new information becomes significant and old information becomes obsolete. These abilities are essential for learning robots.

To demonstrate CS-learning, this article describes the implementation of a CS-learning robot, CS-HERO, that performs the approach-recognize task for a nontrivial set of objects.

For each object, CS-HERO actively selects sensing procedures, moves close to the object if graspable, predicts its appropriate grasping procedure, and executes the grasping procedure to pick it up. This article shows that highly accurate robot classification procedures can be learned without sacrificing efficiency on per object basis. CS-HERO's limitations arise from its assumptions on sensing, moving, and grasping procedures and on its environment:

1. Sensing procedures are noncontact, independent, and fixed routines.
2. Grasping procedures are fixed routines. If a grasping procedure fails to grasp an object, it assumes that the object indeed cannot be grasped by the grasping procedure.
3. The positions of objects can be determined, and objects are reachable.
4. Objects are separated such that the robot is able to sense and grasp one object at a time without any obstruction.
5. The working environment is benign. For example, objects are on a flat surface, there are no hidden or moving obstacles, a sensing value remains true as long as no contact actions are taken, sensing (moving and grasping) costs can be estimated, and there is no emergency (e.g., a low battery).

Future work will include applying CS-learning to other tasks requiring active perception, developing new CS-learning methods, using limited look-ahead search to explore the (cost) dependency among features, and characterizing CS-learning analytically. CS-learning has been applied to address the active perception problem of reinforcement learning (Tan, 1991a).

Acknowledgments

The majority of this work was conducted at Carnegie Mellon University. The author is indebted to Jeff Schlimmer, whose valuable suggestions and assistance made this article possible. I am grateful to Tom Mitchell for his continuous support and discussions. I owe thanks to Jaime Carbonell, John Laird, and Chuck Thorpe for their insightful comments. I thank Rich Sutton, Christopher Matheus, Gregory Piatetsky-Shapiro, and Robert Weihmayer for carefully reading the drafts of this article. Many thanks to Steve Minton and anonymous reviewers for reviewing this article.

Notes

1. This should not be confused with the sophisticted strategy used in ID5 (Utgoff, 1988) that incrementally updates the current tree. We simply assume that the computational costs of building trees are much cheaper than the measurement costs of features.
2. For simplicity of analysis, in this condition feature A and B cost 1.
3. In one case, CS-ID3 appears slightly better than optimal because the data points reflect an average cost over every learning trial, whereas the optimal cost values are calculated only for post-convergence. Averaging over more runs would eliminate this anomaly.

References

Aha, D.W. (1989). Incremental, instance-based learning of independent and graded concept descriptions. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 387–391). Ithaca, NY: Morgan Kaufmann.

Aha, D.W., & Gladstone, R.L. (1990). Learning attribute relevance in context in instance-based learning algorithms. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society* (pp. 141–148). Cambridge, MA: Lawrence Erlbaum Associates.

Breiman, L., Friedman, J.H., Olsen, R.A., & Stone, C.J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth.

Chen, C.H., & Mulgaonkar, P.G. (1990). Feature-utility measures for automatic vision programming. *Proceedings of the 1990 IEEE International Conference on Robotics and Automation* (pp. 1840–1845). IEEE Computer Society Press.

Cox, L.A. (1989). Incorporating statistical information into expert classification systems to reduce classification costs. *Proceedings of the Second International Workshop on Artificial Intelligence and Statistics* (pp. 9.1–9.11). Fort Lauderdale, FL:

Dean et al. (1990). Coping with uncertainty in a control system for navigation and exploration. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 1010–1015). Menlo Park, CA: AAAI Press/The MIT Press.

Feigenbaum, E.A., & Simon, H.A. (1984). EPAM-like models of recognition and learning. *Cognitive Science, 8,* 305–336.

Fisher, D.H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2,* 139–172.

Gennari, J.H. (1989). Focused concept formation. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 379–382). Ithaca, NY: Morgan Kaufmann.

Goldberg, K., & Mason, M.T. (1990). Bayesian grasping. *Proceedings of the 1990 IEEE International Conference on Robotics and Automation* (pp. 1264–1269). IEEE Computer Society Press.

Hong, K.S., Ikeuchi, K., & Gremban, K.D. (1990). *Minimum cost aspect classification: a module of a vision algorithm compiler* (Technical Report CMU-CS-90-124). School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Keller, R.M. (1987). Defining operationality for explanation-based learning. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 482–487). AAAI Press/The MIT Press.

Laird, J.E., Rosenbloom, P.S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning, 1,* 11–46.

Mason, M. (1985). Manipulator grasping and pushing operations. *Robot hands and the mechanics of manipulation*. Cambridge, MA: MIT Press.

Michalski, R.S., & Chilausky, R.L. (1980). Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems, 4(2),* 125–161.

Miller, D.P. (1989). Execution monitoring for a mobile robot system. *Proceedings of the SPIE Conference on Intelligent Control.*

Mitchell, T.M. (1977). Version spaces: a candidate elimination approach to rule learning. *Proceedings of the Fifth IJCAI* (pp. 305–310). Cambridge, MA:

Mitchell, T.M. (1982). Generalization as search. *Artificial Intelligence, 18,* 203–226.

Mitchell, T.M., Keller, R., & Kedar-Carbelli, S. (1986). Explanation-based generalization: a unifying view. *Machine Learning, 1(1),* 47–80.

Norton, S.W. (1989). Generating better decision trees. *Proceedings of the Eleventh IJCAI* (pp. 800–806). Morgan Kaufmann.

Nunez, M. (1991). The use of background knowledge in decision tree induction. *Machine Learning, 6(3),* 231–250.

Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning, 1,* 81–106.

Stansfield, S.A. (1989). Reasoning about grasping. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 768–773). AAAI Press/The MIT Press.

Tan, M. (1991a). Cost-sensitive reinforcement learning for adaptive classification and control. *Proceedings of the Ninth National Conference on Artificial Intelligence.* AAAI Press/The MIT Press.

Tan, M. (1991b). *Cost-sensitive robot learning.* Doctoral dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. (Technical Report CMU-CS-91-134).

Tan, M., & Schlimmer, J.C. (1989). Cost-sensitive concept learning of sensor use in approach and recognition. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 392–295). Ithaca, NY: Morgan Kaufmann (also ref. CMU-CS-89-124).

Tan, M., & Schlimmer, J.C. (1990). Two case studies in cost-sensitive concept acquisition. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 854–860). Menlo Park, CA: AAAI Press/The MIT Press.

Tan, M., & Schlimmer, J.C. (1991). A cost-sensitive machine learning method for the approach and recognize task. *Robotics and Autonomous Systems*, *8*, 31–45. Elsevier-North Holland.

Trinkle, J., & Paul, R. (1989). The initial grasp liftability chart. *IEEE Journal of Robotics and Automation*, 5(1) 47–52.

Utgoff, P.E. (1988). ID5: An incremental ID3. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 107–120). Ann Arbor, MI:

Received January 21, 1992
Accepted August 17, 1992
Final Manuscript October 28, 1992