Robot training through incremental learning

Robert E. Karlsen[a], Shawn Hunt[a], Gary Witus[b]

[a]U.S. Army –TARDEC, Warren, MI 48397-5000
[b]Turing Associates, Ann Arbor, MI 48103

ABSTRACT

The real world is too complex and variable to directly program an autonomous ground robot's control system to respond to the inputs from its environmental sensors such as LIDAR and video. The need for learning incrementally, discarding prior data, is important because of the vast amount of data that can be generated by these sensors. This is crucial because the system needs to generate and update its internal models in real-time. There should be little difference between the training and execution phases; the system should be continually learning, or engaged in "life-long learning". This paper explores research into incremental learning systems such as nearest neighbor, Bayesian classifiers, and fuzzy c-means clustering.

Keywords: robot vision, incremental learning, terrain classification, unmanned ground vehicle

1. INTRODUCTION

There are many applications where a continuous stream of data needs to be analyzed. Examples include video and audio analytics, Internet search, and financial markets. Some applications require real-time analysis of the data because the results have a limited lifetime, while other applications can process the data over hours, days or weeks. Some applications are able to store and have access to all incoming data, while other applications have limited storage capacity and therefore require that the majority of the data must be discarded once it has been processed. Some applications just need an analysis of what is currently happening, while in other cases there is a need to make predictions about future events. In some applications, the algorithm parameters can be determined via an initial training session and then frozen in place, with the algorithm just processing the incoming stream using the pre-determined parameter set. In other applications, the right parameters cannot be determined beforehand and the algorithm is expected to be flexible and adaptable to new inputs that may not have been encountered previously.

In robotics applications there are a number of streams of data that could be analyzed in order to enable autonomous operations, including data streams from video, LIDAR, SONAR, RADAR, GPS, motor currents, force and torque sensors, wheel and joint encoders, steering and throttle commands, and inertial sensors. These data sources could be used for many different types of autonomous operations, employing many types of algorithms. In this work, we are especially interested in those autonomous operations and algorithms where machine learning is employed, such as learning the relationships between sensor inputs and the steering and throttle commands from a human driver, between grasp efficacy and grasp location for the manipulation of objects, or between terrain trafficability and camera inputs.

In looking for suitable learning algorithms for these types of robotic applications, we have a number of desired features that the algorithms should possess. Since our robots typically do not have the capacity to store all sensor inputs, we need an algorithm that can train incrementally; discarding most of the input data after it has been processed. Since the learning will often impact autonomous navigation or manipulation tasks, the processing must be accomplished in near real time. We want the capability to combine knowledge gained from multiple runs of the same robot or from multiple runs of different robots in order to multiply the learning effect. In some cases, the external sensor input will not be sufficient to make reliable predictions, such as when estimating the underlying terramechanics of vegetation-covered terrain or the surface properties or strength of materials to be grasped. Therefore, the algorithm must be capable of quickly adapting to variable conditions that are revealed when physical contact with the objects or environment occur.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
18 APR 2011	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Robot training through incremental learning	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Robert E. Karlsen; Shawn Hunt; Gary Witus	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
US Army RDECOM-TARDEC 6501 E 11 Mile Rd Warren, MI 48397-5000, USA Turing Associates, Annarbor, MI 48103	**21733**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
US Army RDECOM-TARDEC 6501 E 11 Mile Rd Warren, MI 48397-5000, USA	**TACOM/TARDEC/RDECOM**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) **21733**

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
Presented at SPIE 25-29 April 2011 Orlando, Florida, USA, The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**SAR**	**9**	

For this paper, we are using data collected for the purpose of estimating vehicle terrain interaction parameters from visual imagery to test various incremental learning algorithms. To simplify testing, we manually labeled portions of the imagery with terrain classes. Manually labeling data is tedious work and not very useful in general for a realistic system. However, for the purposes of testing learning algorithms, we chose this process to reduce the variance in the output parameters. The terrain trafficability problem is ideally suited for the type of self-supervised learning we have described previously because there are suitable feedback parameters that can be used to guide the learning process.

We will test a number of incremental learners from the WEKA[1] machine learning package on the terrain classification data set, as well as an incremental fuzzy clustering algorithm that we have developed. We compare classification performance, as well as analyze the attributes of the various learning algorithms to see if they meet the criteria that we have established.

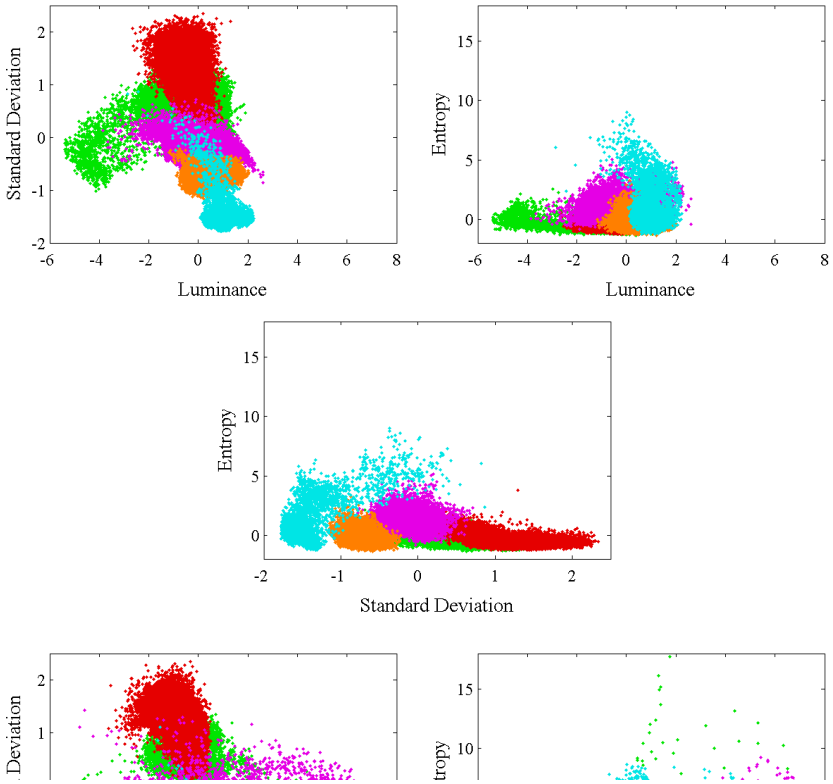
Examples of similar efforts include Hadsell et al.[2], which extends work that was performed under the LAGR program, using stereovision and self-supervised near-to-far learning, to learn both the features and the subsequent terrain classification in an incremental manner, and Kim et al.[3], where on-line terrain classification was performed, using color stereovision features in combination with inertial measurements, drive currents, and other proprioceptive data.

2. DATA

To be useful for implementation on an unmanned system, terrain classes should be based on what mobility the terrain affords to the robot, not on artificial classes of terrain that rely on surface composition. Nonetheless, in this paper, we did use a priori terrain classes in order to isolate the image processing and incremental learning portion of the problem from the vehicle/terrain interaction portion of the problem, which can introduce its own errors. We used video imagery captured on a small robot as it crosses five terrain types: grass, rocks, concrete, sand, and pavers (see Fig. 1).

We divided each image into 12 x 12 patches and computed a variety of features for each patch, including mean and standard deviation of luminance and color, and standard deviation and entropy texture measures at different resolutions. We first applied a Canny edge detector to suppress strong edges before computing the local standard deviation and entropy texture metrics. Features for each image patch were also computed by averaging the features in surrounding patches, rather than by using the features of the subject patch. All together we computed sixty features for each image patch. The input stream was color video, but automatic white balancing and internal color compression artifacts rendered the color features useless, but are still included in the total number of features. Note also that the imagery in Fig. 1 tends to be mostly gray, which may have reduced the usefulness of color anyway. For purposes of this study, three local image features were selected as primary evidence factors: luminance local mean, mean of standard deviation and standard deviation of entropy, computed from the surrounding patches.

The training data set contained 57,100 records, while the test set contained 55,610 records, each with a code for the terrain type and the values of the three evidence factors for that portion of the terrain. The five terrain types were not equally represented, with from 5,000 to 16,000 records per terrain type. The training set was normalized so that each feature had zero mean and a standard deviation of one. The test set was normalized with the same parameters as the training set, but did not have zero mean or standard deviation of one. The two-dimensional scatter-plots in Figs. 2 and 3 show the overlap of the evidence factors for the five terrain types for both the training and testing data sets. The figures show some significant differences between the training and testing sets, which led to some of the errors seen when the learning algorithm attempted to predict the class of the test set.



3. INVESTIGATION OF WEKA ALGORITHMS

WEKA (Waikato Environment for Knowledge Analysis)[1] was used to compare four incremental learning methods. We focused on IB1, IBk[4], naïve Bayes[5], K-star[6], LWL[7], and NNge[8]. The IB1 and IBK algorithms are both derivatives of the nearest neighbor classifier. We chose those four algorithms because they are able to handle data either in batches or incrementally. There are several other incremental algorithms available in WEKA that operate on nominal attributes, but we focused on the algorithms that allowed us to use our dataset without converting the scalar inputs to enumeration type categories. The IB1, Instance Base 1-Neighbor, and IBk, Instance Base k-Neighbors, algorithms are both derivatives of the nearest neighbor classifier. IB1 uses a normalized Euclidean distance measure with 1-nearest neighbor to find the training instance closest to the given test instance. There are no editable properties for the IB1 algorithm within WEKA. The IBk algorithm uses K-nearest neighbors. The basic IB1 and IBk algorithms store every instance of the training data. The WEKA implementation did not pre-filter the training data to add only "informative" new instances and did not prune the instance set to remove highly-similar redundant instances. There are several properties that may be changed for the IBk algorithm: the number of nearest neighbors to use in prediction and how the neighbors will be weighted (either by the inverse of their distance or by their similarity). The K-star algorithm is similar to the IB1 and IBk algorithms, but instead it uses entropy as its distance measure. Locally Weighted Learning, LWL, is another instance-based algorithm that uses locally weighted training to combine training data.

The NNge algorithm, Non-Nested Generalized Exemplars, learns incrementally by first classifying and then generalizing each new example. The generalized exemplars are in a bounded group of instances that share a close proximity within an n-dimensional problem space. This bounding is implemented by using axis-parallel n-dimensional rectangles, which are referred to as hyperrectangles. NNGE was developed as an extension to the Nested Generalized Exemplar (NGE) algorithm[9]. In the NGE algorithm, hyperrectangles were allowed to be nested and overlap. NNge attempted to fix this problem by using non-nested and non-overlapped hyperrectangles.

NNge uses a modified Euclidean distance function that handles hyper-rectangles, symbolic features, and exemplar and feature weights. The class predicted is that of the single nearest neighbor. It uses dynamic feedback to adjust exemplar

and feature weights after each new example is classified. When classifying an example, one or more hyper-rectangles may be found that the new example is a member of, but which are of the wrong class. The NNge algorithm prunes these so that the new example is no longer a member.

The naïve Bayes algorithm is based on Bayes theorem. It assumes conditional independence of the evidence features for each given category. It builds an estimate of the probability of seeing evidence value E_k given class C, for all classes C. The probability of being in class C, given an observed evidence value, is then estimated as the overall fraction of time in class C times the product, over all evidence dimensions k, of the probability of observing evidence value E_k given being in class C. This requires mapping continuous evidence values into the enumerated evidence bins. The incremental naïve Bayes algorithm incrementally updates the probability distribution estimates.

The following experiments were conducted within the WEKA Knowledge Flow environment, a graphical front-end to the algorithms. This environment enables the use of incremental learning algorithms, as opposed to the WEKA Explorer which is for batch learning algorithms. Once a model was trained with training data, the model was modified to no longer be updated from instances from the test datasets.

These experiments were performed on four datasets (as mentioned earlier, there were 57,100 records in the training dataset and 55,610 records in the test dataset). The first dataset selected three of the sixty available features (also mentioned earlier): luminance local mean, mean of standard deviation, and standard deviation of entropy. The second dataset was the same as the first but with the order of the records in the training dataset changed. The third dataset contained all 60 features, and the fourth dataset was the same as the first but with the training and test records swapped. Table 1 shows the error rates for the algorithms we tested. We used 2 nearest neighbors for the IBk algorithm.

	Dataset 1	Dataset 2	Dataset 3	Dataset 4
Naïve Bayes	9.24%	9.24%	17.12%	15.36%
IB1	12.40%	12.40%	8.99%	7.34%
IBk	11.23%	11.23%	8.92%	8.92%
K-star	13.26%	13.26%	9.33%	9.78%
LWL	41.72%	41.72%	40.16%	47.61%
NNge	12.71%	17.14%	9.47%	9.04%

Table 1: The error rates of the algorithms tested within WEKA on the four datasets used in the experiment.

The Naïve Bayes algorithm was not impacted by changing the order of the features in the training dataset, but its performance was decreased when all available features were used and when the training and test sets were swapped. The IB1, IBk, and K-star algorithms were also not negatively impacted by the ordering of the data in the training datasets, but they performed better when all 60 features were used and when the training and test sets were swapped. LWL performed the worst out of the algorithms tested, performing only slightly better than chance. NNge was negatively impacted by changing the ordering of the records in the training dataset, but the performance increased when all 60 features were used, indicating that it does a better job of ignoring irrelevant features. This could be due to NNge's dynamic attribute weighting. NNge maintains a weighting for each attribute that it has seen; the larger the weight, the more important it is. An irrelevant attribute will have values that are sparsely distributed and will be discriminated against. NNGE also performed better when the training and test sets were reversed.

The IB1, IBk, and LWL algorithms store the training data, which is not the ultimate goal of our work. Their results were included in this paper because they could be trained incrementally within WEKA and to provide a comparison against the other algorithms that we tested.

The execution time of the naïve Bayes algorithm performed was the fastest of all of the algorithms we tested. The average training time over the four datasets was 520 ms. The average testing time was 720 ms. The execution time of IB1, IBk, LWL, and k-star algorithms were linear in the size of the training set. NNge had the longest training time of the algorithms, with an average of 98 seconds to train and approximately the same for testing.

4. FUZZY CLUSTERING

We have been exploring the fuzzy c-means (FCM) clustering algorithm[10] for a number of years in the areas of terrain understanding and classification[11,12]. However, the training was always in batch mode. We are now developing an incremental framework using FCM clustering as the base algorithm[13]. We use the simplest FCM variant with spherical clusters of the same size[14]. We have performed some testing using the Gustafson–Kessel variation of the FCM algorithm, which allows the clusters to vary in shape, while maintaining the same volume, but did not find that the classification results warranted the increased complexity[13].

FCM clustering is attractive because it is an unsupervised process, working only with the independent variables, which in this case are the features extracted from the terrain classification imagery. Once the clusters are determined, they can be tagged with a variety of information, such as terrain roughness or resistance, as in our previous work[11], or the terrain class, as in this paper. The independence of the tags and the cluster centers allows the tags to be changed rather quickly if necessary. Also, the unsupervised nature of the algorithm allows different models to be merged together by just concatenating the list of cluster centers. This can be useful for merging models from different days, seasons, environments, or from different vehicles.

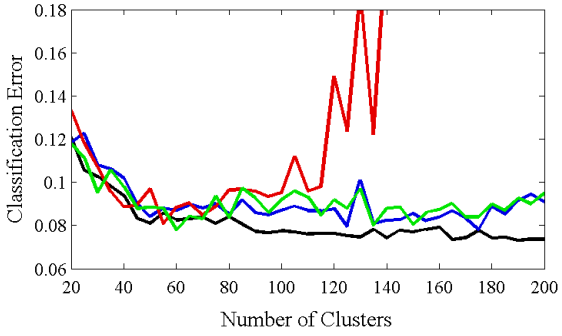


Fig. 3: Incremental learning classification error versus number of clusters for different numbers of synthetic data points (black=batch training, red=2000 pts, green=3500 pts, blue=5000 pts).

Although FCM is an unsupervised algorithm, we have implemented a semi-supervised variant of the algorithm, where the class labels for the training data are used to separate each batch of data into different classes. This was done in order to make sure that each class was afforded an equal number of clusters, since the total number of clusters is fixed in the version of FCM that we are using. We have found in previous work that terrain classes with high variability were assigned most of the clusters when training with the full data set, leading to reduced performance when classifying the test set. We believe that this process can still be carried over to when the goal is to classify terrain based on the mobility performance of the platform, by separating the data into groups with some arbitrary division of the continuous mobility characteristics.

The proposed framework for an incremental FCM algorithm starts with an initial set of clusters, which would of necessity have been generated by a batch process. Statistical measures are computed for each cluster, based on the members of each cluster, after which the raw data can be deleted. As new data arrives, it is classified according to the existing clusters. Data points that fall within the existing clusters are not currently used, since it is not yet clear if it is useful to modify the cluster statistics based on these recognized data samples. Data points that are outside the existing clusters are notionally stored in a buffer until a critical limit is reached. At that point a resampling of the data is performed, where synthetic data is generated, based on the existing cluster statistics, with the cluster members taking on the overall class of the cluster. The buffered unclassified data is added to the synthetic data and the FCM algorithm is run on this combined data set, generating a new set of clusters with associated statistics. The optimum size for the buffer and the number of synthetic data points still needs to be determined from experiment. Our approach of storing and using the cluster statistics differs from that of Hore, Hall and Goldgof[14], where in their approach the clusters are replaced by their centroid, essentially placing all the cluster data at the cluster center.

A number of investigations were performed to test whether the proposed framework would yield satisfactory results. These investigations used the 57,100-sample training set and the 55,610-sample test set described previously. The training data was concatenated into one file and the data sent in segments of various sizes to the online clustering algorithm. Rather than store the unrecognized data in a buffer and waiting for a limit to be reached, the unrecognized data was merged with the generated synthetic data as each segment of data was sent to the clustering algorithm.

There are a number of parameters to choose when converting the batch algorithm to an online algorithm. Two that we examine here are the batch size, i.e. how much data should be analyzed before re-running the clustering algorithm, and the number of synthetic data points, i.e. when generating synthetic data based on the statistics of each cluster, how many total points should one generate.

Figure 3 shows results for the classification error versus the total number of clusters when the number of data points generated by the synthetic data generation algorithm is varied. The training data is separated into 20 segments and sent sequentially to the clustering algorithm. Each feature vector in the segment is analyzed as to whether it is recognized by the system. Those that are not recognized are added to set of N synthetic data points generated from statistics of the clusters computed from the previous segments. The black line is the batch training result using all the training data and no synthetic data generation. The red line corresponds to N=2000 synthetic data points, green to 3500 data points, and blue to 5000 data points. Ultimately, the 57,100 data points from the training set are reduced to N synthetic data points by this process. One can see that the error begins to diverge for the choice of 2000 synthetic data points when more than 100 clusters are specified, whereas the results for 3500 or 5000 data points show no divergence up to 200 clusters and the classification errors are roughly the same. Since the clustering algorithm is faster for smaller numbers of both data points and clusters, choosing the minimum of both, while maintaining comparable performance, would seem to indicate that 2000 synthetic data points with 60 clusters would be a good choice for this data set. However, to have some flexibility in choosing a larger number of clusters, it may be safer to use 3500 synthetic data points. Since the synthetic data generation involves pseudorandom number generation, each of the data points in Fig. 3 is the average of five different runs. The standard deviations of the data points in Fig. 3 range from 0.006 to 0.012.

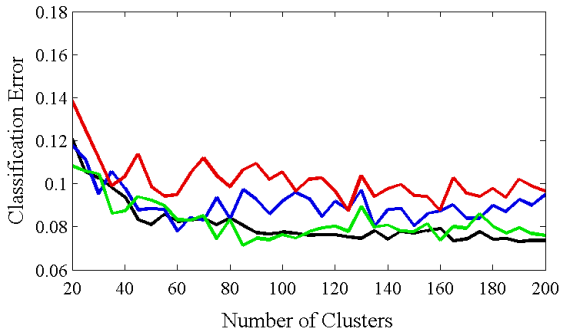
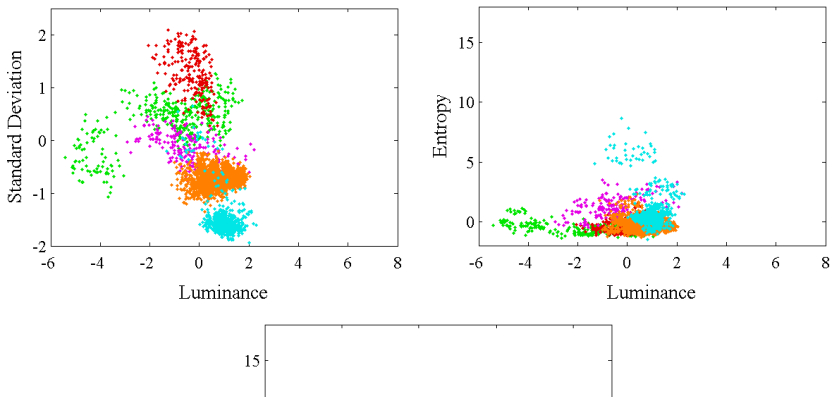


Fig. 4: Incremental learning classification error versus number of clusters for different numbers of data segments (black=batch training, red=30 segments, blue=20 segments, green=10 segments).

Figure 4 shows results for the classification error versus the total number of clusters when the size of the segments of data that are sent to the clustering algorithm is varied. Again, the black line is the batch training result using all the training data (and therefore one segment of data). The green line corresponds to the training data being separated into 10 segments, blue corresponds to 20 segments, and red corresponds to 30 segments. Each of the data runs with more than one segment used 3500 synthetic data points in the analysis. It should also be noted that when sequentially adding data segments, the number of clusters allocated per feature class is kept constant. For example, in the initial portion of the run, where there is only one class in the data set, the number of clusters is specified to be 1/5 the total number shown in Fig. 4, given that we have five total classes specified. By the time all the data has been assimilated, the number of classes is five and the total number of data classes is as shown in the figure. The choice for segment size is somewhat dictated by the desired responsiveness of the system, with larger segment sizes resulting in the system taking longer to respond to new classes. On the other hand, we can see from Fig. 4 that smaller segments can result in less accurate predictions, coming perhaps from the accumulation of errors or from a smoothing effect from the resampling process.



Resampling of the data is an important step in our proposed framework and is performed multiple times. To verify that the resampling is producing data that visually appears to reproduce the structure of the data, in Fig. 5 we plot the synthetic data that resulted from a run with 100 clusters, 3500 synthetic data points, and 10 segments. The synthetic data no longer has zero mean or a standard deviation of one, but appears to match the training set in Fig. 2, albeit much sparser. The test error that results from performing FCM clustering on this particular set of synthetic training data was 0.076.

We performed a test where we changed the order that the data files were presented to the system to see if that would have an effect. From Fig. 6 we can see that it does indeed have an effect and the issue is related to the "rocks" terrain class (file 10), which we have had issues with previously. The blue and cyan lines are from data files where the rocks class was presented first to the system, with the other classes in different orders, while the red and magenta lines are from data files where the rocks class was not presented first. The magenta line corresponds to Data Set 2 in Table 1. The black line is again the batch training result using all the training data. More research needs to be performed to understand the cause of this order dependence and develop methods for correcting for it.

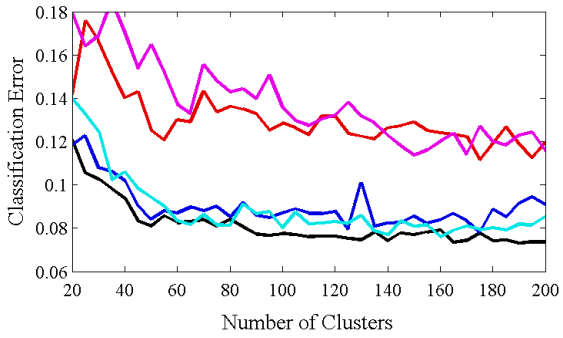


Fig. 6: Incremental learning classification error versus number of clusters for different orderings of training data (black = batch training, red = [37,19,10,28], magenta = [19,10,37,28], blue = [10,19,28,37], cyan = [10,37,28,19]).

In analyzing the synthetic data points generated by our resampling procedure, we noticed that the more recent data seemed to be weighted more heavily than prior data, perhaps resulting in the poorer results when more data segments are chosen or the training order dependence that we have observed. With this in mind, we have been investigating alternative methods for merging the real data with the synthetic data. For example, we have tried clustering the unexplained data first, and then generating synthetic data points, which are added to the synthetic data points from the previous runs. The results of this process are displayed in Fig. 7 for 3500 synthetic data points and 10 data segments, where the green line corresponds to the original way of merging the data and corresponds to the green line in Fig. 4. The red line corresponds to the case where 5 clusters are allocated to each class when clustering the unexplained data and the blue line corresponds to an adaptive number of clusters assigned to each class depending on the number of unexplained data points. More work needs to be performed in determining the parameter settings for the adaptive case and for determining how to handle situations where there are only a few data points that are unexplained in each data segment. In any case, these variations do not appear to change the classification accuracy of the baseline algorithm.

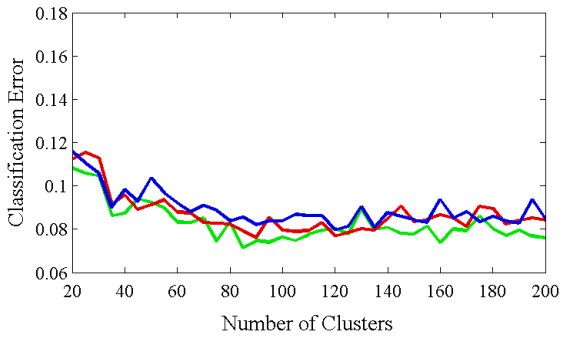


Fig. 7: Incremental learning classification error versus number of clusters for different methods of merging synthetic data (green=no clustering, red=5 clusters, blue=adaptive clusters).

We also switched the training and testing set, corresponding to Data Set 4 in Section 3, and applied the incremental FCM algorithm with 3500 data points and 10 data segments, corresponding to the green lines in Figs. 4 and 7. The results for this are shown in Fig. 8, where the green line corresponds to the normal order of the data and the red line

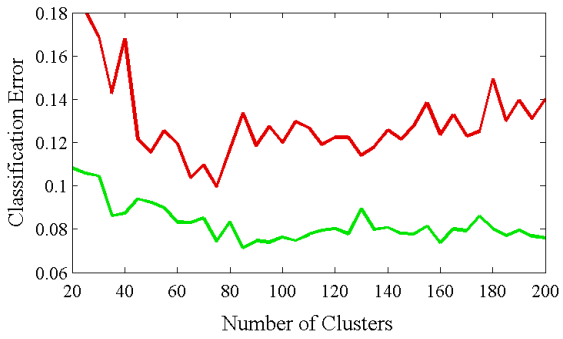


Fig. 8: Incremental learning classification error versus number of clusters (green=normal set, red=training/testing sets reversed.

corresponds to the training and testing data sets being switched. Finally, we also tried to test the incremental FCM algorithm with the whole 60 element data set, corresponding to Data Set 3 in Section 3, but the algorithm did not handle the increased number of features well, both in terms of speed and in classification accuracy.

5. DISCUSSION

In this paper, we have tested a number of incremental algorithms with a data set taken from a terrain classification problem. We compared a number of older algorithms that are part of the WEKA[1] machine learning environment with a fuzzy c-means (FCM) clustering algorithm that we have been exploring in recent years. Based on Figs. 3, 4 and 7, the average error rate for the FCM algorithm on Data Set 1, the baseline data set, was about 9%, which is similar in performance to the Naïve Bayes algorithm and better than the other algorithms in Table 1. The average FCM error rate in Fig. 6 of about 13% on Data Set 2, where some of the data files were presented in a different order, is about the same as the nearest neighbor algorithms, worse than the Naïve Bayes, and better than NNGE. The FCM algorithm did not do well on the full sixty-element feature set of Data Set 3, where the nearest neighbor and NNGE algorithms had good performance compared to the Naïve Bayes. On Data Set 4, where the training and test sets were reversed, the FCM algorithm had an error rate of 13%, which is slightly better than the Naïve Bayes, but worse than the nearest neighbor and the NNGE algorithms. As is usual with classification comparisons, different algorithms perform better on different data sets.

This study was limited in a number of respects. We used a priori terrain classes and manual labeling. Vehicle motion or operator response was not taken into account. The datasets sampled only a limited range of terrains, weather, and illumination conditions. The Naïve Bayes algorithm is known to have issues when the data is not linearly separable, but does work well in many instances and is fast. The nearest neighbor algorithms require that all the data be stored and is therefore not really useful for our robotic applications, but we may explore variations along the lines of the generalized exemplar approach. There are also many other incremental learning algorithms that were not tested, such as Q-learning reinforcement learning and incremental Hidden Markov Models. Future work includes testing incremental Support Vector Machines, such as the open source code released by Cauwenberghs[16] and also other variations of the NNge algorithm[17].

REFERENCES

[1] I. Whitten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition, Morgan Kauffman, San Francisco (2005).

[2] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller and Y. LeCun, "Learning long-range vision for autonomous off-road driving," *J. Field Robotics* **26**(2) 120-144 (2009).

[3] D. Kim, J. Sun, S.M. Oh, J.M. Rehg and A.F. Bobick, "Traversability classification using unsupervised on-line visual learning for outdoor robot navigation," Proc. 2006 IEEE Int. Conf. Robotics and Automation (ICRA), 518–525 (2006).

[4] D. Aha, D. Kibler and A. Albert, "Instance-based learning algorithms," *Machine Learning* **6**, 37-66 (1991).

[5] J. George and P. Langley, "Estimating continuous distributions in Bayesian classifiers", Proc. 11th Conf. Uncertainty in Artificial Intelligence, 338-45 (1995).

[6] J. Cleary and L. Trigg, "K*: An Instance-based learner using an entropic distance measure," Proc. 12th Int. Conf. Machine Learning, 108-14 (1995).

[7] C.G. Atkeson and A.W. Moore, "Locally weighted learning," *Artificial Intelligence Review* **11**(1), 11-73 (1997).

[8] B. Martin, “Instance-based learning: nearest neighbor with generalization,” Master’s Thesis, University of Waikato, Hamilton, New Zealand (1995).

[9] S. Salzberg, “A Nearest Hyperrectangle Learning Method”, *Machine Learning*, 6(3):251-276 (1991).

[10] F. Hoppner, F. Klawonn, R. Kruse and T. Runkler, *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition,* John Wiley & Sons (1999).

[11] R.E. Karlsen and G. Witus, “Terrain perception for robot navigation,” Unmanned Ground Vehicle Technology IX, SPIE Proc. 6561, 65610A (2007).

[12] R.E. Karlsen and G. Witus, “Adaptive learning applied to terrain recognition,” Unmanned Ground Vehicle Technology X, SPIE Proc. 6962, 69620H (2008).

[13] R.E. Karlsen and G. Witus, “Incremental learning and perception,” 27th Army Science Conference, Orlando FL (2010).

[14] B. Balasko, J. Abonyi and B. Feil, “Fuzzy clustering and data analysis toolbox,” (2008). (www.fmt.vein.hu/softcomp/fclusttoolbox).

[15] P. Hore, L.O. Hall and D.B. Goldgof, “Single pass fuzzy c means,” Proc. IEEE Int. Conf. Fuzzy Systems, London. (2007).

[16] G. Cauwenberghs and T. Poggio, “Incremental and decremental support vector machine learning,” *Adv. Neural Information Processing Systems*, Cambridge, MA: MIT Press, Vol 13 (2001).

[17] A. Wong, “Investigating noise tolerance in generalized nearest neighbor learning,” Honor’s Report, University of Canterbury, New Zealand (2005).