Automatic hierarchical classification using time-based co-occurrences

Abstract

While a tracking system is unaware of the identity of any object it tracks, the identity remains the same for the entire tracking sequence. Our system leverages this information by accumulating joint co-occurrences of the representations within the sequence. These joint co-occurrence statistics are then used to create a hierarchical binary-tree classification of the representations. This method is useful to classify sequences as well as individual instances. We illustrate the use of this method on two separate representations- the tracked object's position, movement, and size; and the tracked object's binary motion silhouettes.

1 Introduction

Infants have an innate ability to do primitive tracking. As they track objects, they are unaware of the identity of the object. But as long as they observe a stable input signal, they can be relatively certain that the object is the same object. Whether this information is simply useful in the development of a normal visual system or absolutely necessary can be debated at another time. In either case, we are investigating using the information gained by simple, primitive tracking behavior to aid in visual tasks- in particular, unsupervised hierarchical classification.

1.1 Our approach

Our simple adaptive background tracker, as described in [1, 4], has tracked over 10 million objects over the past 15 months. As shown in Figure **??**, every frame that an object is tracked its location (x,y), speed/direction (dx,dy), and size are recorded. Also, an image of the object and a binary motion silhouette are cropped from the original image and the binary difference image respectively. The two sets of experiments covered in this paper perform classification based on the {x,y,dx,dy,size} representation and the binary motion silhouette representation using literally millions of training examples. Rather than using sequences to create a sequence classifier(as is most common), we are using the sequences to create a instance classifier.

Our method involves developing a codebook of representations using an on-line Linear Vector Quantization(LVQ) on the entire set of representations acquired by the tracker. Second, we accumulate joint co-occurrence statistics over the codebook by treating the set of representations in each sequence as an equivalency multi-set. Finally, we perform hierarchical classification using only the accumulated co-occurrence data.

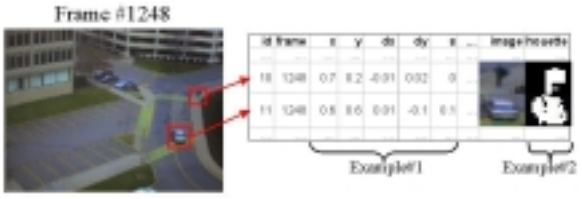


Figure 1: *This figure shows a single frame from a typical scene and the information which recorded for the two moving objects. The fields which are used for the two classification examples are labeled.*

1.2 Previous work

Our method is most similar to the work of Johnson and Hogg [2]. They begin their process by on-line Linear Vector Quantization on the input space. But, they then quantize again into a predetermined number of probability distribution functions (pdfs) over their discrete states. While a significant number of these pdfs will result in tight clusters of activity, it is unclear how to relate two inputs that are grouped into separate pdfs or to select the proper number of pdfs.

Our hierarchical classification involves a step that has the flavor of Normalized Cuts and its many derivatives (see [3]). It has discrete nodes (defined by the codebook). It has edges which represent pair-wise distances (or dissimilarities or costs) between them. In addition, the goal is to determine two sets of nodes that are dissimilar. However, that is the extent of the similarity. Our "costs" are probabilities, not "distances." Those similarities are not directly related to the coordinates or properties of the nodes, but rather are measured empirically from the data. Our "cut" does not produce two discrete sets that minimize the cut "similarities." It produces two distributions that both explain the observed joint statistics and are relatively dissimilar and match the co-occurrence data.

The following sections describe the method, show two sets of results, discuss ways of improving this method, and draw conclusions.

2 The Method

This section outlines the basic process. First, a codebook is generated using Linear Vector Quantization. Second, the automatically tracked sequences are used to define a co-occurrence matrix. Finally, the co-occurrence data is used to probabilistically break apart the representations in the codebook into a binary tree representation.

We assume that the tracker will produce sequences of representations of the same object(e.g. a particular person or particular vehicle). This must be taken into account when developing the tracker, because accidental tracking errors can result in false equivalences.

2.1 Codebook generation

A codebook is a set of prototypes which can be used to represent the input. Once a codebook is generated, it is used as a lookup table for incoming values. Given the desired size of the codebook, the goal of quantizing is to determine a set of prototypes which best represents the dataset. Our results were produced with codebooks of 400 prototypes. An input is defined by the prototype which is nearest to it. More complex spaces (e.g. color image space) would necessitate a either more prototypes or more complex prototypes.

Depending on the complexity of the input space, it may be difficult to create an effective codebook of representations. If all the representations in the codebook are equally likely to result from all the underlying classes, this system will fail. For example, if none of the representations in your codebook is more likely to result from a person than a vehicle, there will be no possibility of using those representations to differentiate people and vehicles without additional information.

While this may seem unsettling, we are encouraged by our ability to generate large codebooks. Large codebooks are usually troublesome because as the size of the codebook, K, increases, the amount of data needed for effective codebook generation increases on the order of K. Also, the amount of data needed for co-occurrence statistics accumulation increases on the order of K^2. Since our system automatically collects and processes data, we have hundreds of gigabytes of tracking data for future processing steps. And, our method converges as the amount of data increases rather than suffering from over-fitting.

For the quantity of data and the number of prototypes we use, an off-line method, such as K-means, is not an option. The simplest method of Linear Vector Quanitization is to initialize the codebook randomly with K prototypes centered at existing data points. Then, take a single data point and find the closest prototype in the codebook. Then, adapt that prototype towards the current data point using a learning factor, α. The α value is slowly decreased over time until the prototypes are static.

Unfortunately, a prototype may be too far from any data points to "win" leaving it stranded in an area where it doesn't represent significant data. We have circumvented this problem with a method used by Johnson and Hogg[2].

An area of high data point density may accumulate a large portion of the prototypes, leaving few prototypes for the rest of input space. In some cases, it may be desirable to have a large number of prototypes in the high-density areas because those regions may be the most ambiguous regions of the input space (e.g. traffic at an intersection). In other cases, the areas of high density may arise from uninteresting, repetetive input data (e.g. scene clutter) and there is no benefit to wasting a large portion of your prototypes in that region. We currently filter most of the sequences which are less than a few seconds in duration. This filters most of the repetative motions in the scene before the learning process.

2.2 Accumulation of co-occurrence statistics

Once the codebook has been generated, the input space is no longer considered. Every input data point is labeled as the most representative prototype- the one that is nearest to it. So rather than considering a sequence of images, binary silhouettes, positions, or histograms, we only consider sequences of symbols, s_1 through s_K, corresponding to the K prototypes.

Further, our method disregards the order of the sequence and considers them as multi-sets of symbols. A multi-set is a set which can contain multiple instances of the same element.

The goal of this system is to produce a classification system which can be given one or more observation(e.g. an image, a silhouette, etc.) of a particular class and classify it. This is in contrast to systems that are specifically designed to recognize sequences (e.g. Hidden Markov Models). When the system has learned to classify an object based on its motion silhouette, color histogram, or size, it should be capable of doing so with a single example. Of course, the system should perform better if given multiple examples, but it should not rely on seeing a complete sequence.

Our model for the production of the sequences is simple. There are N underlying classes which occur

with some prior probability, π_c. A class c, when observed, has some probability distribution, $p_c(.)$, defining the probability it will produce each of the prototype's symbols. As long as the object is observed, it will produce symbols given the same distribution. This model reflects our assumption of the independence of samples in a sequence discussed earlier.

The multi-sets of prototypes are used to estimate a co-occurrence matrix, C where $c_{i,j}$ is the estimated probability that a sequence from the training sequences will contain an input represented by the i^{th} prototype and a separate input represented by the j^{th} prototype.

First, an matrix of the accumulated co-occurrences, $C_{i,j}^{total}$, is initialized to zeros or a prior joint distribution (see Future work section). Given a multi-set, each possible pair (excluding pairing symbols with themselves) is added to C^{total} weighted inversely by the number of pairs in that sequence. Given a sequence, $S = \{S_1, S_2, ...\}$, for each pair $\{S_i, S_j \; where \; i \neq j\}$

$$C_{i,j}^{total} = C_{i,j}^{total} + 1/P \tag{1}$$

where $P = |S|^2 - |S|$ is the number of valid pairs in this sequence. Then the current joint co-occurrence estimate, C, is C^{total} normalized

$$C = C^{total}/Z \tag{2}$$

where Z is the number of sequences currently used to estimate C^{total}.

If there was a single underlying class and infinite sequences to train, $C_{i,j}$ would converge to $p_1(i) * p_1(j)$. In such a case, nothing can be said about the relative relationships of the prototypes. With N underlying classes,

$$\lim_{Z \to \infty} C_{i,j} = \sum_{c=1}^{N} \pi_c * p_c(i) * p_c(j) \tag{3}$$

Given enough synthetically produced data from a system for which each class has one prototype for which it is the sole producer, it is possible to solve for all parameters of the model. Since this is a restrictive case, we will not pursue it here. The next section outlines how our system extracts a hierarchical approximation to these classes.

2.3 Hierarchical classification

Our classification method involves taking the entire set of prototypes and the co-occurrence matrix and attempting to determine two distributions, or probability mass functions(pmfs), across the prototypes of the codebook that best explain the co-occurrence matrix. Once these distributions are determined, each distribution is treated as another set of prototypes and their co-occurrence matrix is estimated. The process is repeated to produce a binary classification tree of a predetermined height.

The root of the tree represents the universal pmf including every prototype in proportion to how often it actually occurred. At each branch in the tree, the pmf is broken into two separate pmfs that are relatively dissimilar. This process does not necessarily guarrantee that the two pmfs will sum to the parent pmf.

At each branch, we initialize two random pmfs with two priors, π_1 and π_2, use the pmfs and priors to create an estimate of the co-occurrence matrix,

$$\hat{C}_{i,j} = \sum_{c=1}^{N} \pi_c * p_c(i) * p_c(j) \tag{4}$$

and iteratively re-estimate the parameters to minimize the sum squared error

$$E = \sum_{i,j} (C_{i,j} - \hat{C}_{i,j})^2 \tag{5}$$

The update rules are

$$\pi_c = (1 - \alpha_\pi) * \pi_c + (\alpha_\pi) * \sum_{i,j} (C_{i,j} - \hat{C}_{i,j}) * p_c(i) * p_c(j) \tag{6}$$

and

$$p_c(i) = (1 - \alpha_p) * p_c(i) + (\alpha_p) * \sum_{j} (C_{i,j} - \hat{C}_{i,j}) * p_c(j) \tag{7}$$

where c is the class($c \in \{0, 1\}$) and the learning factor for the priors, α_π, is higher than the learning factor for the pmfs, α_p. It is sometimes useful to put soft constraints on the priors to insure both distributions represent significant portions of the co-occurrence data.

At each branch, the parent distribution is used to estimate the co-occurrences that result from that class. The co-occurrence for the left branch subproblem would be derived from the original co-occurrence, C, and the left pmf, $p_0(.)$ as follows

$$C_{i,j}^0 = Ci, j * p_0(i) * p_0(j) \tag{8}$$

C^0 is used to determine the children pmfs of $p_0(.)$, $p_{00}(.)$ and $p_{01}(.)$. For example, if a pmf was uniform over half the prototypes, the co-occurrence matrix used for its children would include only the co-occurrences between those prototypes. If this was not done, every branch may result in the same pmfs as the initial branch.

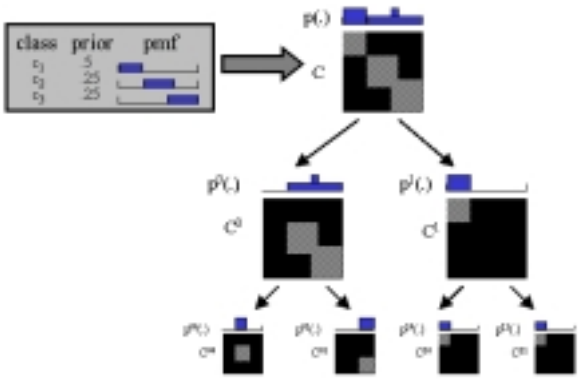


Figure 2: *This figure shows a synthetic classificaton example with three underlying classes. The first branch separates the class which doesn't have any overlap from the other two. That separable class cannot be further separated. The other two classes are separated at the next branch. Increasing the height of this synthetic example would result only in duplicating the current leaf nodes.*

2.4 Using the classifier

Once the parameters for the pmfs have been determined. Any exclusive set of them can be used as classifiers. An exclusive set of prototypes can be determined by using the leaf nodes of any pruned version of the binary tree. In the examples below, we have simply chosen a single level to do the classification.

Each observation in a sequence is treated as an independent observation. Thus the probability of a particular class is the product of the probabilities of that class producing each of the observations in the sequence. This can be computed by using the dot product of the log of the pmfs(with prior) with the accumulated prototype histogram from the sequence.

2.5 A simple example

Figure 2 shows a synthetic example. Using the predefined classes and priors, a root pmf and a root co-occurrence matrix can be formed. At each branch the pmf is broken into two pmfs which best explain the observed joint co-occurrences. The classification hierarchy behaves as would be expected, first breaking apart the class which never presents like the other two classes, then breaking the other class.

3 Results

The following two examples involve creating a classification hierarchy using the same number of prototypes, the same learning parameters, and the same sequences produced by our tracking system. The only difference is that one uses position, speed, direction, and size as a representation while the other uses binary motion silhouettes as a representation.

3.1 Classifying activities

This example classifies objects based on a representation of their position, speed, direction and size (x,y,dx,dy,s). First, four hundred representative prototypes are determined. Each prototype represents all the objects of a particular size that are seen in a particular area of a scene moving in a particular direction. Co-occurrences are accumulated using 24 hours of sequences from that scene. Finally, the universal pmf (the true pmf of the entire set of sequences) is probabilistically broken into two pmfs.

The process is repeated to produce a binary tree of height 4 detailed in Figure 3. Figure 4 shows the history of one particular day.

3.2 Classifying motion silhouettes

While this example results in a rather simple classification, it illustrates an intended use for this type of classification. LVQ resulted in 400 silhouettes of vehicles and people. The first break broke the silhouettes into two relatively discrete classes, people and vehicles. Some of the more blurry prototypes remained ambiguous because they matched both vehicles and people. Figure 5 shows the co-occurrence matrix, the pmfs, and some examples of prototypes from both classes. In this case, the first branch was the only significant branch. Further, branches were less interesting. The resulting classifier was over 98people.

4 Future work

The most obvious weakness of this algorithm is the need to discretize complex input spaces. We are currently investigating automatically deriving local feature sets using LVQ on sub-images and learning those features similarities using local (in time and space) co-occurrence measurements. Doing this hierarchically hold promise for learning useful feature sets and better prototypes.

This could also be useful for texture segmentation. For example, create 10,000 texture prototypes and define their similarity based on which prototypes occur near other prototypes (spatially and temporally). **Learning** similarities this way, rather than attempting to assert a prior for which textures are similar, takes advantage of domain specific regularities and could define regularities in domains where it is not certain how similar two textures are.

Of course, assumed similarities are useful, particularly in cases where there is not enough data. In such cases, the C^{total} can be seeded with a co-occurrence matrix. Hence, prototypes without sufficient representation will assume the similarities they are given while

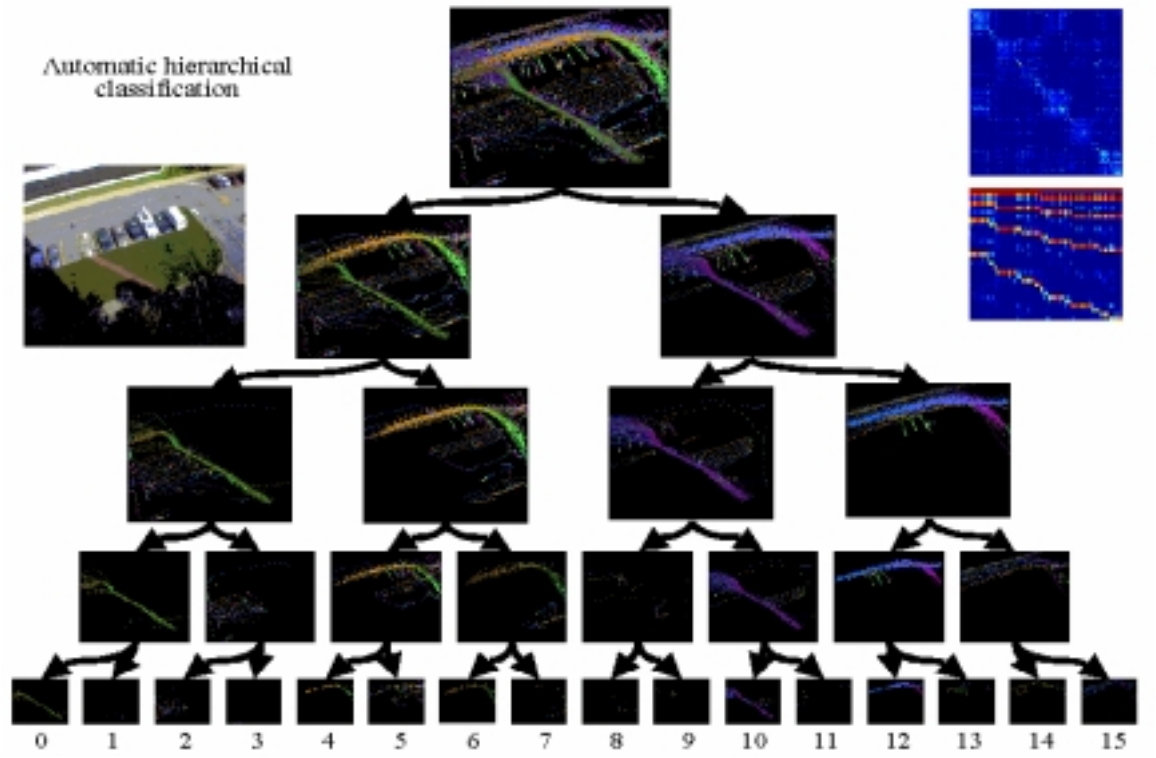
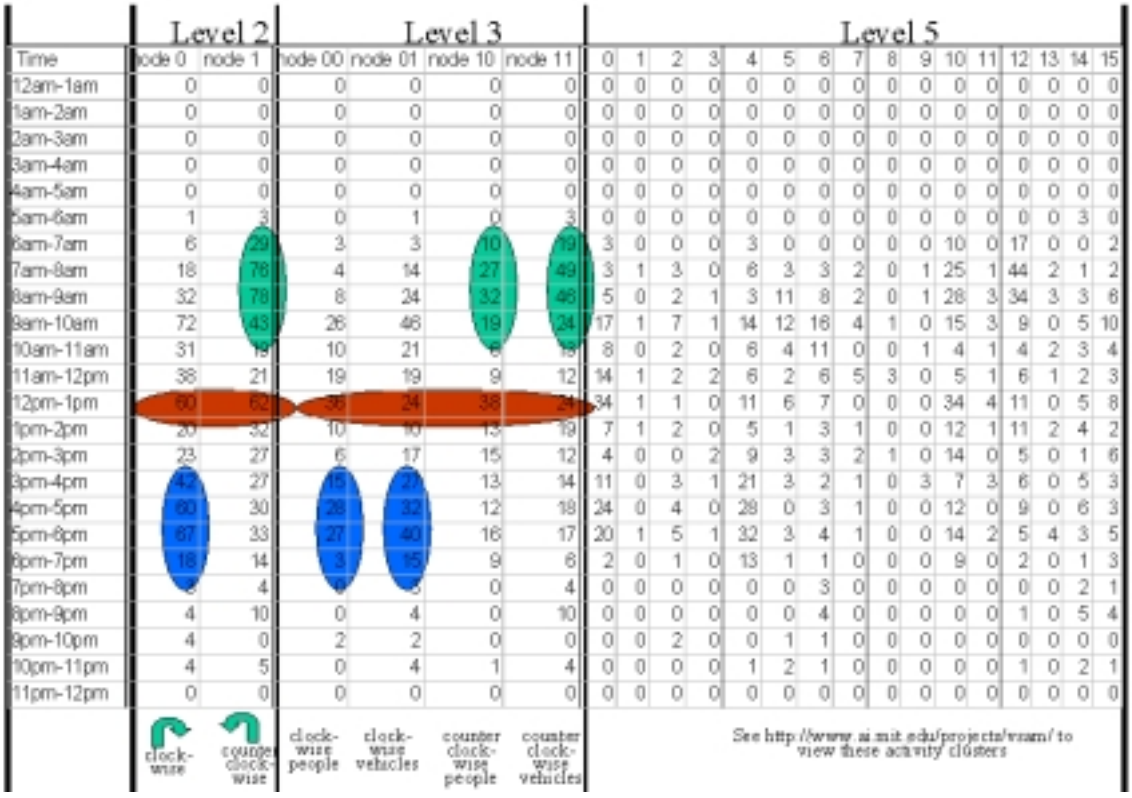


Figure 3: *This figure shows an image of the scene(upper left), the classification hierarchy(center), and the co-occurrence matrix(upper right) and normalized pmfs for each element of the tree. The scene contains a road with adjacent parking spots and a path through the grass near the loading bay of our building. The binary tree shows accumulated motion templates for each node of the tree. The first break separates traffic moving in one direction around the building and traffic moving in the other direction, because objects in this scene did not generally change their direction. The second break for both branches separates traffic on the road and traffic on the path. While there are some prototype states which we common to both activities, these two activities were significantly different and accounted for a significant amount of the data. Further bifercations result in classes for: pedestrians on the path; pedestrians and lawn-mowers on the lawn; activity near the loading dock. cars; trucks; etc. These classes can be viewed in a Java 1.1 compatible browser at: http://www.ai.mit.edu/projects/vsam/Classification/Cclasses/. Note: the co-occurrence matrix has been ordered to make some of its structure more evident.*

Figure 1: Classification counts for 9/21/98



	Level 2		Level 3				Level 5													
Time	node 0	node 1	node 00	node 01	node 10	node 11	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
12am-1am	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1am-2am	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2am-3am	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3am-4am	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4am-5am	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5am-6am	1	3	0	1	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
6am-7am	6	29	3	3	10	19	3	0	0	0	3	0	0	0	0	0	10	0	17	0	0	2
7am-8am	18	78	4	14	27	49	3	1	3	0	6	3	3	2	0	1	25	1	44	2	1	2
8am-9am	32	78	8	24	32	46	5	0	2	1	3	11	8	2	0	1	28	3	34	3	3	6
9am-10am	72	43	26	46	19	24	17	1	7	1	14	12	16	4	1	0	15	3	9	0	5	10
10am-11am	31	19	10	21	6	13	8	0	2	0	6	4	11	0	0	1	4	1	4	2	3	4
11am-12pm	38	21	19	19	9	12	14	1	2	2	6	2	6	5	3	0	5	1	6	1	2	3
12pm-1pm	60	62	36	24	38	24	34	1	1	0	11	6	7	0	0	0	34	4	11	0	5	8
1pm-2pm	20	32	10	10	13	19	7	1	2	0	5	1	3	1	0	0	12	1	11	2	4	2
2pm-3pm	23	27	6	17	15	12	4	0	0	2	9	3	3	2	1	0	14	0	5	0	1	6
3pm-4pm	42	27	15	27	13	14	11	0	3	1	21	3	2	1	0	3	7	3	6	0	5	3
4pm-5pm	60	30	28	32	12	18	24	0	4	0	28	0	3	1	0	0	12	0	9	0	6	3
5pm-6pm	67	33	27	40	16	17	20	1	5	1	32	3	4	1	0	0	14	2	5	4	3	5
6pm-7pm	18	14	3	15	9	6	2	0	1	0	13	1	1	0	0	0	9	0	2	0	1	3
7pm-8pm	8	4	4	4	0	4	0	0	0	0	0	0	3	0	0	0	0	0	0	0	2	1
8pm-9pm	4	10	0	4	0	10	0	0	0	0	0	0	4	0	0	0	0	0	1	0	5	4
9pm-10pm	4	0	2	2	0	0	0	0	2	0	0	1	1	0	0	0	0	0	0	0	0	0
10pm-11pm	4	5	0	4	1	4	0	0	0	0	1	2	1	0	0	0	0	0	1	0	2	1
11pm-12pm	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	clock-wise	counter clock-wise	clock-wise people	clock-wise vehicles	counter clock-wise people	counter clock-wise vehicles	See http://www.ai.mit.edu/projects/vsam/ to view these activity clusters															

Figure 4: *This figure shows how many of the activities were detected on a particular day. The first two columns correspond to the initial branch. The following four columns correspond to the next level of the binary classification tree. The last 8 columns are the leaf nodes of the classification tree. Below some of the columns the primary type of activity for that node is listed. Morning rush hour is highlighted in green and shows traffic moving mostly in one direction. The lunch-time pedestrian traffic is highlighted in red. The evening rush hour is highlighted in blue and shows more movement in the opposite direction as the morning rush hour.*

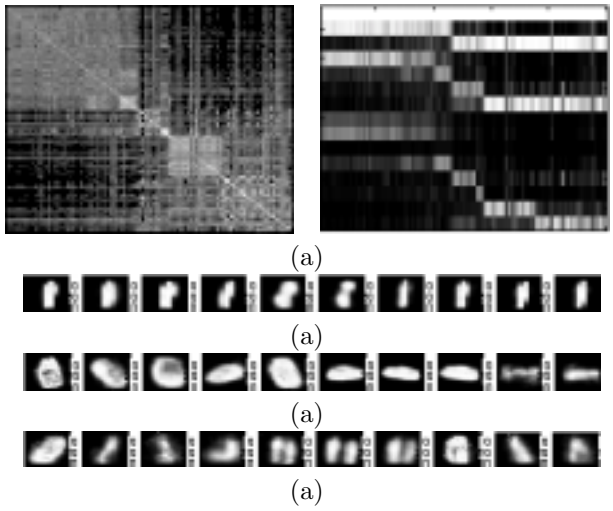


Figure 5: *(a) shows the co-occurrence matrix and resulting pmfs. Some of the prototypes from the person class(b), vehicle class(c), and some prototypes which were significantly ambiguous(d). In C, the upper left corresponds to silhouettes of people and the lower right corresponds to silhouettes of vehicles. The vehicles show less statistical independence because vehicles in this particular scene were only scene as they passed through particular orientations. If the scene contained vehicless driving in circles, the corresponding prototypes would exhibit more independance. Note: the co-occurrence matrix has been ordered to make some of its structure more evident.*

the similarities of the prototypes which are observed often are determined by the data.

Finally, we are investigating using both the prototypes and the co-occurrences to detect outliers. If many data points in a sequence are not represented by a prototype, it may be an unusual event. Also, a sequence's co-occurrences are very unlikely given the joint co-occurrences, it is likely to be unusual.

5 Conclusions

We have motivated and implemented a new approach to automatic object classification. This approach has shown promise with two contrasting classification problems. In one case, it produced a non-parametric activity classifier. In the other case, it produced an binary image-based classifier. We are currently investigating many other possible uses for this method.

Acknowledgments

This section is intentionally left blank for the review process.

References

[1] Grimson, W.E.L., Chris Stauffer, Raquel Romano, and Lily Lee. "Using adaptive tracking to classify and monitor activities in a site," In *Computer Vision and Pattern Recognition 1998(CVPR98)*, Santa Barbara, CA. June 1998.

[2] Johnson N. and Hogg D. C. "Learning the Distribution of Object Trajectories for Event Recognition." In Pycock D., editor, *British Machine Vision Conference*, pages 583-592. BMVA, September 1995.

[3] Shi, Jianbo and Jitendra Malik. "Normalized Cuts and Image Segmentation," In *Proc. of the IEEE Conf on Computer Vision and Pattern Recognition*, San Juan, Puerto Rico, June 1997.

[4] xxx, xxx. "Adaptive background mixture models for real-time tracking," In submission, 199x.