Analysis of Simulated Crowd Flow Exit Data: Visualization, Panic Detection and Exit Time Convergence, Attribution, and Estimation

Anna Grim, Boris Iskra, Nianqiao Ju, Alona Kryshchenko, F. Patricia Medina, Linda Ness, Melissa Ngamini, Megan Owen, Randy Paffenroth, and Sui Tang

Abstract This paper describes the results of exploratory analyses of black box simulation data modeling crowds exiting different configurations of a one-story building. The simulation data was created using the SteerSuite platform. Exploratory analysis was performed on the simulation data without knowledge of simulation algorithm. The analysis effort provided a hands-on introduction to issues in crowd dynamics. Analyses focused on visualization, panic detection, exit convergence

A. Grim
Brown University, Providence, RI, USA
e-mail: anna_grim@brown.edu

B. Iskra · F. P. Medina (✉) · R. C. Paffenroth
Worcester Polytechnic Institute, Worcester, MA, USA
e-mail: biskra@wpi.edu; fpmedina@wpi.edu; rcpaffenroth@wpi.edu

N. Ju
Harvard University, Cambridge, MA, USA
e-mail: nju@g.harvard.edu

A. Kryshchenko
California State University of Channel Islands, Camarillo, CA, USA
e-mail: alona.kryshchenko@csuci.edu

L. Ness
Rutgers University, New Brunswick, NJ, USA
e-mail: linda.ness@rutgers.edu

M. Ngamini
Morehouse College, Atlanta, GA, USA

M. Owen
Lehman College, City University of New York, Bronx, NY, USA
e-mail: megan.owen@lehman.cuny.edu

S. Tang
Johns Hopkins University, Baltimore, MD, USA
e-mail: stang@math.jhu.edu

E. Gasparovic, C. Domeniconi (eds.), *Research in Data Science*,
Association for Women in Mathematics Series 17,
https://doi.org/10.1007/978-3-030-11566-1_11

pattern discovery, identification of parameters influencing exit times, and estimation of exit times. A variety of mathematical and statistical methods were used: k-means clustering, principal component analysis, normalized cut grouping, product formula representation of dyadic measures, logistic regression, auto-encoders, and neural networks. The combined set of results provided insight into the algorithm and the behavior modeled by the algorithm and revealed the need for quantitative features modeling and distinguishing the shapes of the building configurations.

1 Introduction

This paper describes results of exploratory analyses of black box simulation data modeling crowds exiting different configurations of a one-story building. Exploratory analysis was performed on the simulation data without knowledge of simulation algorithm. The analysis effort provided a hands-on introduction to issues in crowd dynamics research and more generally provided a hands-on introduction to analysis of agent-based data generated by an unknown algorithm.

We gratefully acknowledge Mubbasir Kapadia for permitting use of the SteerSuite platform [35] to provide the data for our research effort and Weining Lu for designing and executing the simulation scenarios and for providing and documenting the data. This paper is dedicated to Weining Lu who tragically died shortly after we began the analysis of this data set.

Crowd dynamics, or pedestrian dynamics, is an area of research that covers a wide range of approaches related to understanding and modeling crowd behavior. The most practical motivation for understanding crowd dynamics is to improve human safety in real-world crowd situations. For example, in 2015 at least 2411 pilgrims were killed in a stampede during the Hajj pilgrimage at Mecca [15], and in 2017 there were multiple crowd stampedes resulting in injury and death (e.g., [4, 24]). Better understanding and modeling of how crowds behave can also be used to improve crowd flow during emergency situations requiring evacuations of buildings, sports stadiums, airline and rail terminals, and large public spaces. Advances in crowd behavior research can potentially improve tracking of people's movements from real-world crowd flow data. Finally, crowd dynamic models are used to produce realistic computer-generated crowds in video games and movies.

One main area of research in crowd dynamics is methods for modeling realistic crowd behavior [20], either to generate realistic simulations for testing other factors, such as building safety, or for use in video games and movies. Modeling crowd behavior is almost always approached using agent-based modeling [8]. In an agent-based model of crowd dynamics, individual people or groups of people are represented by agents, who are given a set of rules and properties, possibly all the same or differing by agent, for how the agent should interact with its environment and the other agents. Methods for formulating the rules to steer pedestrians include: ego-centric fields [21] and social force models [17].

Another problem is how to create crowd heterogeneity in a simulation. Traditionally, this involves a lot of customization, such as tweaking the features of individual agents, to achieve the desired effect [34]. However, an active area of research is how to reduce this customization work, such as by making the agent behavior activity-centric, through the use of "influences" which encode agent desires [23].

A final problem in crowd dynamics simulation is developing the software to execute the proposed agent-based modeling system. Recently, the program Menge [10] was created as an extensible framework for simulating pedestrian movement in a crowd. It allows users to either create their own plug-ins to solve different subproblems of the crowd dynamics simulation problem or to use built-in solutions for those subproblems that are not being explicitly tested. We used data generated by the recently developed SteerSuite platform, designed to be an open framework for developing, evaluating, and sharing steering algorithms [17, 20, 21, 35].

An alternative to the agent-based modeling or kinetic modeling approach to simulating crowd dynamics is to treat the crowd as a continuum flow, as in fluid dynamics. This is most appropriate for large-scale, dense crowds. It is also possible to consider both kinetic and continuum, or microscopic and macroscopic, elements, as part of a multiscale crowd model. For example, this has been done using models based on optimal transport [27], and mean field games [25], as well as in modeling crowd emotions [9, 38].

Recently, the detection of abnormal behavior in crowd has attracted a lot of attention in computer vision [36]. In computer vision, the data sets are very large and consist of video streams recorded by surveillance systems such as Closed Circuit cameras installed in the streets, shopping complexes, temples, stadiums, etc. Computer Vision research focused on the techniques of extracting useful feature data from the video stream, and used them to detect abnormal behaviors [30]. The most frequently used features for crowd abnormal behavior include global flow-based features and local spatiotemporal-based features (see [30, 36, 37] and references therein). Our research in this paper is most closely related to this line of research. In the simulation data set we used, the position data of the agents was available, as were a list of features for each agent. We focused on visualizing and inferring the nature of the agent trajectories and on inference of the crowd flow exit dynamics.

2 Paper Overview

The paper is organized as follows. First, in Sect. 3 the simulation scenarios and simulation datasets are described. Briefly, each simulation run data set consists of trajectory locations, time, and parameter settings for 100 agents who are each trying to exit one of nineteen different building configurations. Next in Sect. 4, the initial exploratory analysis experiments are described. Since the analysis was conducted without knowledge of the simulation algorithm, the goal of each experiment was to visualize summaries of the trajectory behavior and acquire intuition about the impact of the initial conditions and key parameters. The trajectories were each

viewed as a vector in high-dimensional space clustered into a few clusters using the most popular clustering algorithm (k-means). The trajectories were then color coded and graphed in Fig. 4. A sparser visual summary was obtained by using the widely used Principal Components Algorithm to find the values of the two-dimensional linearly uncorrelated vectors (two most principal components) which explain the most variance of Fig. 5 trajectory vectors. These exploratory experiments revealed that the trajectories were piecewise linear. The last three experiments all focused on characterization of exit behavior, since we did know that the goal of the simulation algorithm was to attempt to produce exit trajectories for randomly placed agents with different initial assignments of their parameters. In Sect. 5, the goal was to algorithmically identify the agents who exhibit the panic behavior of losing their direction in high-density areas near the exit and circling around the exit rather than moving efficiently toward the exit. This was accomplished by viewing the agents as a similarity graph, with one node for each agent and weighting the edges between the agents using similarity of their trajectories as defined by the Frobenius norm of the difference of their trajectories. In Sect. 6, the goal was to quantitatively and visually compare the rates of convergence to the exit of the whole group of agents among the different room configuration scenarios. The location of the group at each point in time was viewed as a counting measure, which was uniquely characterized by a vector of dyadic product coefficient parameters. The rate of convergence was represented by the time series of distances to the exit location. Four different exit convergence patterns emerged (see Fig. 10). In Sect. 7, logistic regression was used to determine features which influence the probability of escaping from the room. In Sects. 8 and 9, the exit times for the agents are estimated using several different methods: principal component analysis, auto-encoders, and neural networks. The first of these two sections introduces the methods, and the second of these two sections describes the application of the methods. Section 10 summarizes the results and proposes some future research directions. The building configurations are described in Appendix.

3 Description of the SteerSuite Simulation Data Set

The data set consisted of simulated trajectories of 100 randomly placed agents for 19 different configurations of a one-story building. The goal of the simulation algorithm was to steer the agents to exit the building. Each building configuration was simulated approximately 20 times. Most, but not all of the agents succeeded in reaching the exit by the end of the simulation run. The data for each run consisted of 23-dimensional vectors for each agent at each time step. The twenty-three features were: agent id, time, x and y coordinates for position and velocity, goal and final target (the exit), the radius, acceleration, personal space threshold, agent repulsion importance, query radius, body force, agent body force, sliding friction force, maximum speed, two other features for nearby agents, and two wall

Table 1 Features and their description for agents and their trajectories

Trajectory features	Description of features
ID	Agent's ID
Time	Timestamp
Position x	x coordinate of current position
Position y	y coordinate of current position
Velocity x	x coordinate of current velocity
Velocity y	y coordinate of current position
Target x	x coordinate of final target
Target y	y coordinate of final target
Agent features	
Radius	Radius of the agent
Acceleration	The inertia related to mass
Personal space threshold	The distance between a wall and an agent within which a repulsive force begins to act
Agent repulsion importance	The factor which decides how much the penetration depth affects both the repulsive force and frictional force between two agents
Query radius	Defines the area, in which all objects act force on the subject agent.
Body force	Factor of repulsive force between an agent and a wall
Agent body force	Factor of repulsive force between two agents
Sliding friction force	Factor of frictional force
agent_b	The proximity force between two agents is agent_a * EXP($-d*$ agent_b), where d is the closest distance between two agents' outlines
agent_a	agent_b * EXP($-d*$ agent_a)
wall_b	The proximity force between an agent and a wall defined by wall_a * EXP($-d*$ wall_b), where d is the closest distance between two agent's outline and a wall
wall_a	wall_b * EXP($-d*$ wall_a)
Maximum speed	The maximum speed of an agent

parameters. Table 1 shows a brief description of the semantics of the features that was provided. No additional documentation of the algorithm was provided. This provided a realistic exercise in analysis of data generated by an unknown algorithm. Figures for each of the room configuration scenarios were provided. Building configurations for Scenarios 2, 3, and 10 are shown in Figs. 1, 2, and 3.

Each building configuration had 3 rooms on the north side, 2 rooms on the east side, and 2 rooms on the south side. The walls and exits for these rooms were the same for all of the room configurations. Each room configuration was configured with obstacles: 3 bars, 4 rectangular boxes, and 2 square boxes. The placement of the obstacles varied with the room configuration. The coordinates of the obstacles were

Fig. 1 Building configuration 2

not included in the data set. Figure 20 in Appendix section provides an overview of all of the building configurations and is accompanied by a description of the variation among the configurations.

4 Visualizing the Data

We performed several different exploratory analyses on the data, including k-means clustering and principal components analysis (PCA). K-means clustering is widely used to determine if the data can be divided into groups or clusters, while PCA is used to reduce the dimensionality of the data. A detailed description of PCA is given in Sect. 8.2. All code for this section was written in Python. We used the Pandas library [28] for reading in, storing, and manipulating the data. Clustering and PCA was done using the Sci-kit Learn library [33], which works with Pandas' dataframes, and plotting was done using the Matplotlib library [19].

Fig. 2 Building configuration 3

The first experiment was to cluster the trajectories. A trajectory initially consisted of the x and y coordinates of the agent at each time step, arranged as a vector. After an agent exited the room, their trajectory stopped. As agents exited the room at different times, the initial trajectory vectors were of different lengths. We extended these vectors to all be the length of the longest one by adding values equivalent to the position of the exit for the remaining, missing time steps.

We performed k-means clustering on these augmented trajectories, using $k = 5, 6$, where k is the number of clusters to find. We tried several other values of k, but $k = 5, 6$ gave the best results qualitatively. The clustered trajectories are shown in Fig. 4. Trajectories in the same cluster often begin near each other, and have a similar shape. This is not surprising as the trajectories are represented by a sequence of points along them, and k-means clustering uses the Euclidean distance between these vectors as the distance between trajectories. Short trajectories have a lot of identical padding at the end of their vectors, all of which will contribute 0 to the Euclidean distance. This explains why one of the clusters consists of the shortest

Fig. 3 Building configuration 10

trajectories. These clustering results suggest that the trajectory data behave in a predictable way, and thus can be used in more sophisticated data analysis methods.

We further explored the idea that trajectories can be represented by their starting points using principal components analysis (PCA) [18, 32]. We converted the trajectories into vectors using the same method that we used for clustering, and found the first two principal components of these trajectory vector data points. The vectors were then projected into the 2-dimensional subspace given by these principal components, with the result shown in Fig. 5. The vectors are colored by the y-coordinate of the agent's starting position, and the size is proportional to the radius of the agent. Notice that the vectors are still roughly in the same order under PCA. The two PC dimensions explain 91.4% of the variation in the data. This suggests that we could represent the trajectory that an agent takes using their starting point in analyses involving the agent's other features.

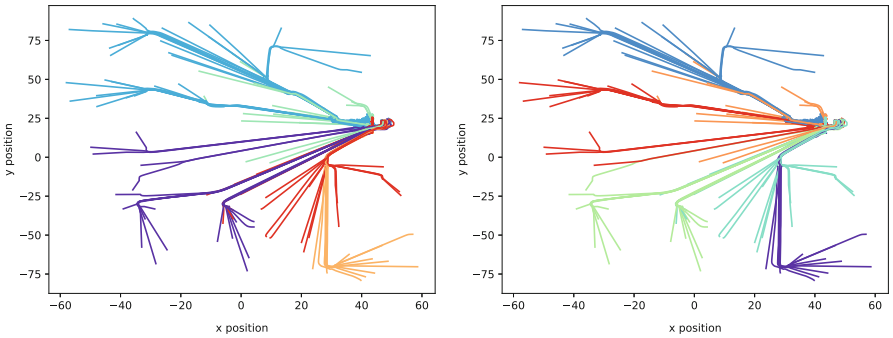

Fig. 4 The trajectories for scenario 10, run 0 colored by cluster, with (**a**) 5 and (**b**) 6 clusters total

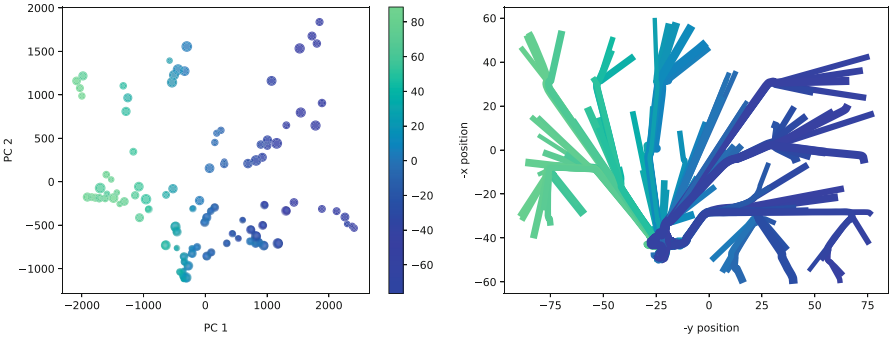

Fig. 5 (**a**) Each point represents one of the trajectories from scenario 20, run 20. The size of the point is proportional to the radius of the agent, and the color of the point corresponds to the y-coordinate of the agents starting position. (**b**) For comparison, the actual trajectories are given in the orientation matching that of (**a**), also colored by their starting y-coordinates

Finally, we plotted each trajectory colored by the time at which the corresponding agent exited the room. See Fig. 6. In general, it took longer for agents further from the door to exit. From the figure, the radius of the agent does not appear to affect the exit time as much as the starting position.

5 Panic Detection in Crowd Dynamics

We considered the panic detection problem from the crowd trajectory data. As we can see from Fig. 7, which visualizes the movements of agents, it is very clear that agents can be approximately clustered into two groups in terms of their behavior: peaceful agents and panic agents. Peaceful agents know where to go almost all along

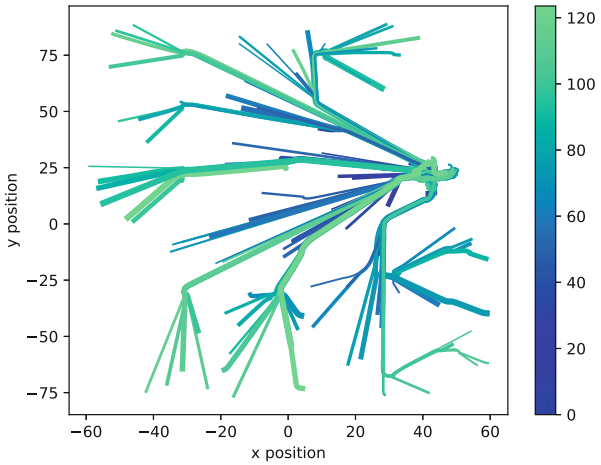

Fig. 6 The trajectories for scenario 20, run 20 colored by the agents' exit times. The thickness of each trajectory is proportional to the radius of the agent

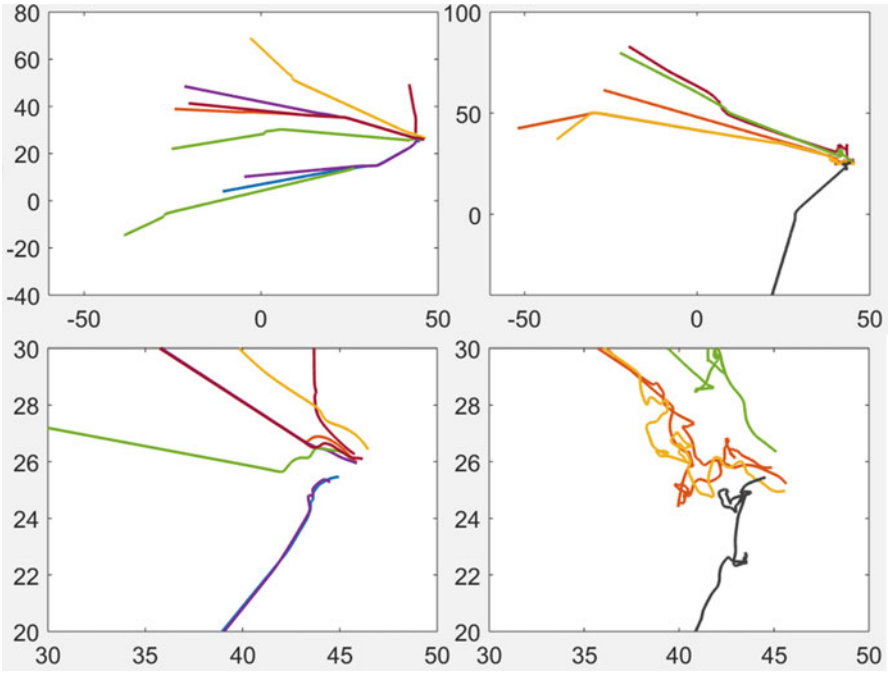

Fig. 7 The trajectory data are taken from scenario 10, run 1. The left top figure displays some typical trajectories for peaceful agents and the right top figure displays some typical trajectories for panic agents. We zoom in on the trajectories near the final exit in the left bottom figure and right bottom figure

the way, so their trajectories are approximately piecewise linear and have very few jumps. Panicked agents lose their direction in the places where the density of agents near them is high, causing them to circle around to find the exit. Informally, by peaceful agents, we mean the agents who can find the fastest route to the exit from their original positions. The others are called panicked agents. Figure 7 displays typical trajectories of peaceful agents and panicked agents in a room and a zoomed-in picture near the final exit. Our goal is to develop an algorithm that can cluster our trajectory data into two groups: one group exhibiting peaceful behavior, and the other group exhibiting panicked behavior.

We interpret this problem as a clustering problem. Suppose we identify each agent with a point in a high-dimensional ambient vector space, with entries consisting of its position at different time instances. Our task is to group these points into two clusters, one consisting of points corresponding to panicked agents, and the other one consisting of points corresponding to peaceful agents. Our idea to perform this specific clustering was based on the classical normalized cut algorithm [26], which clusters a weighted graph into two subgraphs minimizing the connection between them. To do this, we organized the trajectory data for each agent into a matrix $\mathbf{s}_i = \big(x_i(t_k), y_i(t_k)\big)_k \in \mathbb{R}^{M\times 2}$. We define a similarity between agent i and agent j by the Frobenius norm of the difference of their trajectories:

$$W_{ij} = \|s_i - s_j\|_F = \sqrt{\sum_{k=1}^{M} |x_i(t_k) - x_j(t_k)|^2 + |y_i(t_k) - y_j(t_k)|^2},$$

where $\|\cdot\|_F$ denotes the Frobenius norm. In this way, we construct a fully connected undirected graph $\mathscr{G} = (V, E, W)$ with symmetric weights, where the nodes V represent the N agents, and E consists of the edges $\{(i, j) : i, j = 1, \cdots, N\}$ with a weight W_{ij} defined on edge (i, j) measuring the similarity between the agent i and the agent j. We give intuition for why this metric is good for our panic detection. In this evacuation situation, we only have one exit. Our trajectory data of crowds consists of discrete positions at certain time instance. Starting from the same position, the peaceful agents are able to find the fastest route to the exit, while the panicked agents probably lost their direction and cannot find the right way or take a much longer time to find the exit. Near the exit, the peaceful agents (though their origin may be different) know how to move to the unique exit effectively, while the panicked agents are more likely to circle around and approach the exit very slowly. So, the distance between the agents at all time instance is a good candidate for providing the similarity between agents in terms of behavior.

For a vertex $v_i \in V$, the degree of v_i is defined as:

$$d_i = \sum_{j=1}^{n} W_{ij}.$$

Then, the *degree matrix* D is defined as the diagonal matrix with degrees $d_1, \cdots, d_n$ on the diagonal. Given a subset of vertices $A \subset V$, we measure the "size" of A by $vol(A) = \sum_{i \in A} d_i$. For two not necessarily disjoint sets $A, B \subset V$, we define the communication between them by:

$$C(A, B) = \sum_{i \in A, j \in B} W_{ij}.$$

Given a similarity graph with adjacency matrix W, we would like to find a partition of $V = A_1 \cup A_2$ which minimizes

$$Ncut(A_1, A_2) = \frac{C(A_1, A_2)}{vol(A_1)} + \frac{C(A_1, A_2)}{vol(A_2)}.$$

This problem is the so-called *normalized cut problem* with clusters equal to 2. However, it is NP hard and the classical solution is to solve its relaxation which results in the spectral clustering problem. The algorithm for solving this problem is summarized in the algorithm section.

There are other types of graph we can construct such as the ϵ-neighborhood graph and k-nearest neighbor graphs [26]. To our knowledge, how the choice of the similarity graph influences the clustering result remains an open question. For more details about the normalized cut problem and spectral clustering, we refer the readers to [26].

The algorithm is shown in the display captioned "Algorithm 1: Normalized cut grouping algorithm."

Algorithm 1 Normalized cut grouping algorithm [26]

Require: Trajectory data $\{s_i : i = 1, \cdots, N\}$.
Ensure: Two clusters A_1 and A_2
1: Construct a weighted graph by computing weight on each edge and then place data into W and D as described in Sect. 5.
2: Build a normalized Laplacian $L = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ and compute the right eigenvector u that corresponds to the second smallest eigenvalue.
3: Bipartition the graph into two groups $A_1 = \{i : u(i) \geq 0\}$ and $A_2 = \{i : u(i) < 0\}$.

The results of the classification algorithm are shown in Fig. 8.

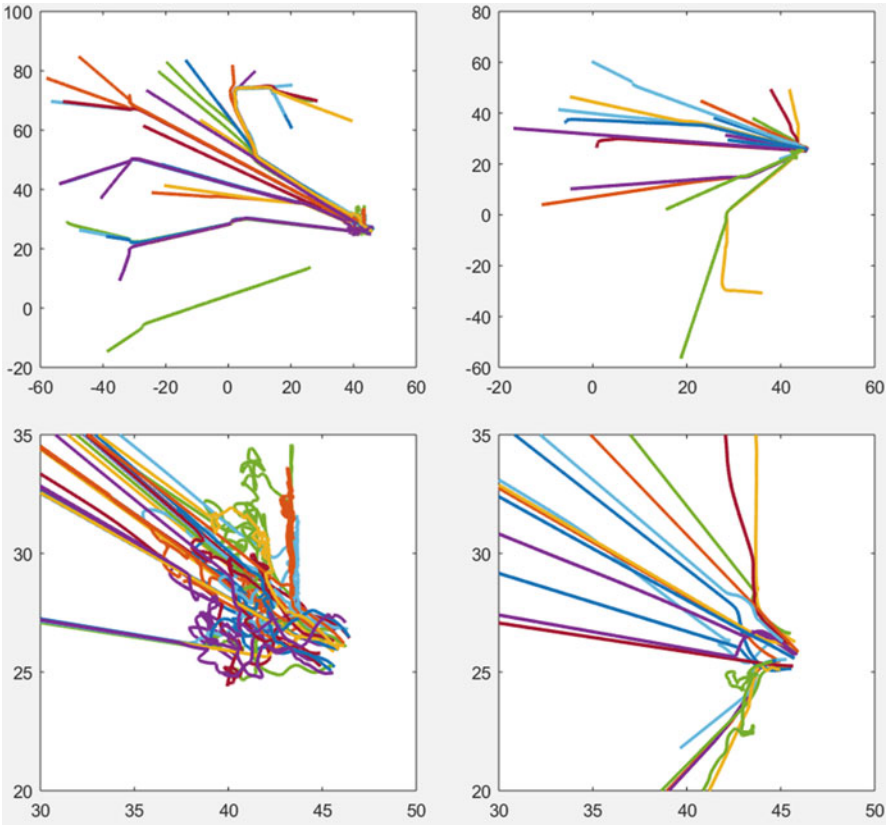

Fig. 8 We apply the clustering algorithm to trajectory data obtained from scenario 10, run 2. In the left top figure, we display the trajectories for cluster 1, and its zoomed-in picture near the exit is displayed in the left bottom figure. In the right top figure, we display the trajectories for cluster 2, and its zoomed-in picture near the exit is displayed in the right bottom figure. As we can see, cluster 1 mainly consists of panic agents and cluster 2 mainly consists of peaceful agents. This example demonstrates the effectiveness of our clustering algorithm

6 Exit Convergence Patterns

In addition to analyzing the trajectories of the agents, we sought to quantitatively characterize the sequence of spatial distributions of the agents for each simulation run. Since the goal of the agents is to exit the room, we expected that each of these sequences of spatial distributions would converge to the delta distribution concentrated at the exit. We also expected that rate and pattern of convergence to the exit delta distribution would vary depending on the configurations of the rooms. The spatial distributions of agents at each step of a simulation can be quantified as a counting measure on the rectangle containing the room. The measure of a subset of the room rectangle is the number of agents in the subset. Each of

these counting measures has the same total measure, since each simulation run has the same number of agents. The Dyadic Product Formula Representation Lemma explained in the next subsection implies that each of these measures is characterized by a unique vector of normalized multiscale parameters. We characterized the rates of convergence by computing the distances between the vector of parameters for the simulation step measures and the exit delta distribution. Ideally, we would then have been able to attribute the differences in rates of exit convergence to differences in the room configurations. However, this last step requires future research because we lacked the information necessary to quantitatively characterize the room configuration.

6.1 The Dyadic Product Formula Representation

As it was proposed in [31], we are viewing each sample (here the data for each simulation run) as a parametrized measure. We use the dyadic product formula representation as in [13] for positive measures in a dyadic set to represent a set of sample data[1] from the dyadic set. We also use it to represent the confidence distribution of the vectors of the product coefficient parameters.

We first recall that a dyadic set is a collection of subsets structured as an ordered binary tree (e.g., unit interval, feature sets, and unit cubes). More precisely, we consider a *dyadic* set X which is the *parent* set or root of the ancestor tree of a system of left and right *child* subsets. For each subset S (dyadic subset) of X, we denote the left child by $L(S)$ and the right child by $R(S)$. Let μ be a nonnegative measure on X and dy the naive measure, such that $dy(X) = 1$.

$$dy(L(S)) = \frac{1}{2}dy(S), \quad dy(R(S)) = \frac{1}{2}dy(S)$$

Note that μ is additive in the binary set system, i.e., $\mu(S) = \mu(L(S) \cup R(S)) = \mu(L(S)) + \mu(R(S))$ (where $L(S)$ and $R(S)$ are disjoint).

Definition 1 Let μ be a dyadic measure on a dyadic set X and S be a subset of X. The *product coefficient* parameter a_S is the solution for the following system of equations:

$$\mu(L(S)) = \frac{1}{2}(1 + a_s)\mu(S) \tag{1}$$

$$\mu(R(S)) = \frac{1}{2}(1 - a_s)\mu(S) \tag{2}$$

[1] In our application, the sample consisted of the positions of the agent at a particular time.

A solution for (1)–(2) is unique if $\mu(S) \neq 0$. If $\mu(S) = 0$, we assign the zero value to the product coefficient, i.e., $a_S = 0$. Note that if $\mu(S) > 0$, then solving (1)–(2) for a_s gives

$$a_s = \frac{\mu(L(S)) - \mu(R(S))}{\mu(S)} \tag{3}$$

The product coefficients are bounded, $|a_S| \leq 1$. In what follows, we use a Haar-like function h_S defined as:

$$1 \text{ on } L(S),\ -1 \text{ on } R(S),\ \text{and } 0 \text{ on } X - S. \tag{4}$$

The product formula for nonnegative measures in $X = [0, 1]$ using the product factors a_S first appeared in [13]. We present below the representation lemma for dyadic sets extracted from [31].

Lemma 1 (Dyadic Product Formula Representation) *Let X be a dyadic set with binary set system B whose non-leaf sets are B_n.*

1. *A nonnegative measure μ on X has a unique product formula representation:*

$$\mu = \mu(X) \prod_{S \in B_n} (1 + a_S h_S)\, dy \tag{5}$$

   where $a_S \in [-1, 1]$ and a_S is the product coefficient for S.
2. *Any assignment of parameters a_S for $(-1, 1)$ and choice of $\mu(X) > 0$ determines a measure μ which is positive on all sets S on B with product formula:*

$$\mu = \mu(X) \prod_{S \in B_n} (1 + a_S h_S)\, dy \tag{6}$$

   whose product coefficients are the parameters a_S.
3. *Any assignment of parameters a_S from $[-1, 1]$ and choice of $\mu(X) > 0$ determines a nonnegative measure μ with product formula:*

$$\mu = \mu(X) \prod_{S \in B_n} (1 + a_S h_S)\, dy \tag{7}$$

   The parameters are the product coefficients if they satisfy the constraints:

   a. *If $a_S = 1$, then the product coefficient for the tree rooted at $R(S)$ equals 0.*
   b. *If $a_S = -1$, then the product coefficient for the tree rooted at $L(S)$ equals 0.*

Example 1 (Formula for a Scale 0 Dyadic Measure) Let $X = [0, 1]$ and let there be a nonnegative measure μ such that $\mu(X) = 1$, $\mu(L(X)) = \frac{1}{4}$, and $\mu(R(X)) = \frac{3}{4}$. Let $a = a_X$ be the product coefficient which is the solution for the system of equations:

$$\mu(L(X)) = \frac{1}{2}(1 + a)\mu(X) \tag{8}$$

$$\mu(R(X)) = \frac{1}{2}(1 - a)\mu(X). \tag{9}$$

Subtracting (9) from (8), we obtain $a = \dfrac{\mu(L(X)) - \mu(R(X))}{\mu(X)} = -\dfrac{1}{2}$.

Since, $dy(X) = 1$ and $dy(L(X)) = \frac{1}{2} = dy(R(X))$ then by the product formula from Lemma 1:

$$\mu = \mu(X)(1 + ah)dy, \tag{10}$$

where h is the Haar-like function as in (4) with $S = X$.

6.2 Analysis of Exit Convergence

For the analysis of exit convergence, we viewed each simulation run of 100 agents exiting a particular room scenario as a sample of an unknown 2-dimensional stochastic process. The only information provided to us about this stochastic process was that it was implemented using a deterministic exit path-planning algorithm, and the initial location and velocity of each agent was assigned randomly for each simulation run. The simulation run data consisted of agent ids, locations at each time step, and values of 20 other features (see Table 1 for definition of the features). Our informal hypothesis was that each of the 19 room configurations together with the average static agent parameters for the simulation runs effectively determined a different stochastic process for exiting the room. Our goal was to characterize differences and similarities between these stochastic processes using unsupervised multiscale representation algorithms. We exploited the product formula method [13, 31] for representing measures on spaces with a binary tree structure to represent the locations of the 100 agents at each time step as a 2-dimensional counting measure. The 2-dimensional space was the rectangle containing the room, and the binary tree structure consisted of the subsets of the room obtained by repeatedly dividing it vertically and horizontally. The same method was exploited to represent the delta distribution measure determined by 100 agents at the exit. At each time step, an $L1$ distance between the vector of parameters for the agent location measure and the vector of parameters for the exit delta measure was computed. For each simulation run, these distances could be viewed as a time series characterizing the

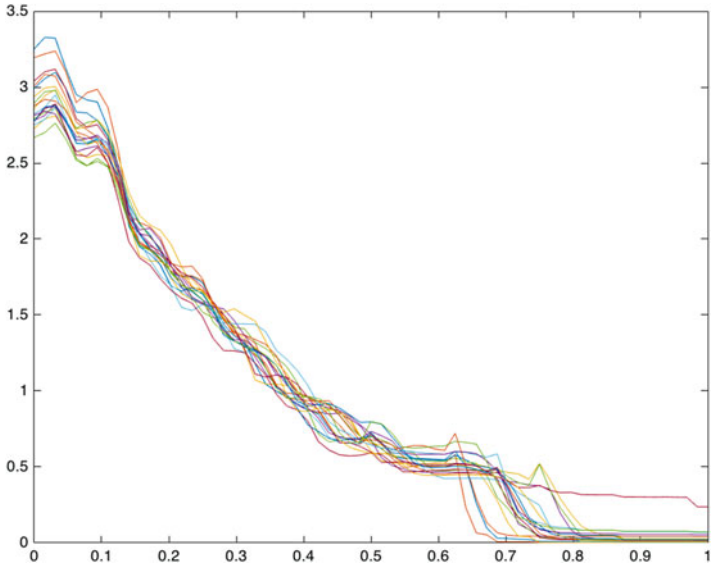

Fig. 9 Average convergence rate time series for the 19 building configuration stochastic processes

rate of convergence of each process to the exit delta measure. We viewed each of these convergence rate time series as a 1-dimensional measure on a space with binary tree structure and used the same method to represent them as a vector of parameters. Here, the space was the maximum time interval for all of the simulation runs, and the binary tree structure consisted of the dyadic sets obtained by repeatedly dividing into halves the interval and resulting subintervals. The measure at the finest scale consisted of the average step function for the time series. The mathematical theory for the product formula method guarantees that measures can be averaged, since averaging the vectors of parameters for a set of measures results in a vector of parameters for a measure. The average convergence rate time series measures for each building configuration were computed by averaging the parameters of the approximately 20 convergence rate time series for each building configuration. Then, the product formula representation lemma was used to compute the average convergence rate time series step function from the parameters. The graphs of these 19 times series step functions were compared. These graphs are shown in Fig. 9. The right-hand portion of the 19 graphs differ and reveal that the 19 average convergence rate times series fall into four groups. Figure 10 shows the group averages of the average convergence rate time series. From left to right in Fig. 10, the Group Numbers are 1, 2, 3, and 4.

Group 1 contains building configurations 2, 3, and 10. Group 2 contains building configurations 7, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, and 20, Group 3 contains building configurations 4, 5, and 6 and Group 4 has just one building configuration 8 (see Fig. 20 in Appendix). The simulation runs for Building Configuration 8

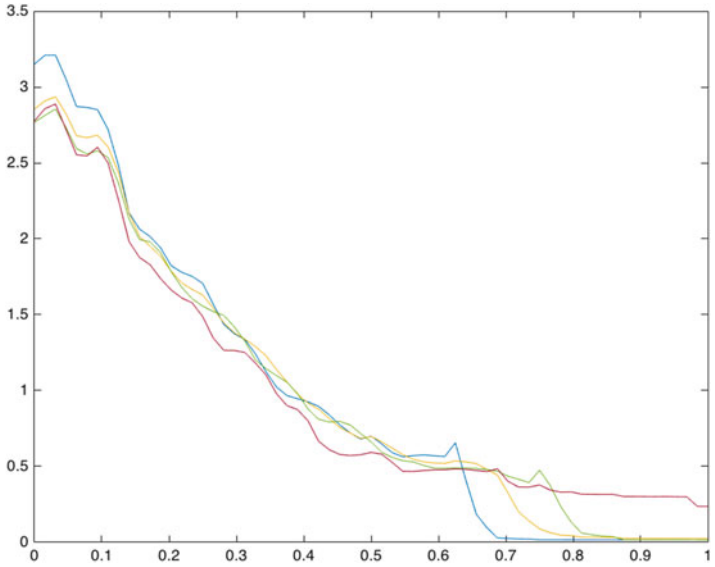

Fig. 10 Average convergence rate time series for 4 groups of building configuration processes

were distinguished by having the most agents still wandering at the end of the simulation run—from 1 to 24, with an average of 7.4. All of the other simulation runs had 1 or occasionally no wandering agents at the end of the simulation run. Group 4 (Building Configuration 8) had the largest average body radius, largest average acceleration, and the largest agent proximity force feature. Group 3 had the lowest average body radius, the lowest average acceleration, and the highest agent proximity force. Group 1 had the highest average sliding friction force.

The building configuration for the only member of Group 4 is shown in Fig. 11.

Group 1 exhibited the most rapid average convergence behavior. The three building configurations for Group 1 are shown in Figs. 1, 2, and 3.

This unsupervised analysis identifies groups with different convergence behavior and qualitatively identifies some differences in the averages of the static parameters for the agents in the simulation runs. Future analysis should attempt to statistically attribute the differences in the groups to agent parameters and to quantitatively characterize the different building configurations, e.g., as summarized in Table 8.

7 Parameters Influencing Probability of Escape

In this section, we will explore the relationship between agent features (independent variables) and binary dependent variable of escaping the room. The goal of this section is to determine which features influence the ability of agents to escape the room once the simulation is finished. We then use these features to build a model that could be used to predict if the agent can escape the room based on the values

Fig. 11 Group 4: building configuration 8—most wandering agents, most incomplete convergence

of its features. Since the outcomes of our desired model are binary, i.e., "escaped" or "not escaped," we will use logistic regression which is a classical method when dealing with binary dependent variables. In fact, logistic regression estimates the probability of an agent escaping the room which then can be used to cluster the agents. For example, we can choose a cutoff value and if the estimated probability is higher than that value we will classify that agent as "escaped," otherwise the agent will be classified as "not escaped."

We defined the escape from the room as 1 if at the final step the agent is one step or less away from the exit of the room and 0 otherwise:

$$Escape_i = \begin{cases} 1, & \text{if } x_{final} \geq 46, y_{final} \geq 24 \\ 0, & \text{otherwise} \end{cases} \quad \text{where } i = 1, \ldots, 100.$$

We consider the Euclidean distance from the starting position of an agent to the exit as a parameter together with the agent features shown in Table 1.

Denote X to be a matrix of data, where the columns represent features and rows represent the agents. Also, let $p_i = P(\text{Escape}_i = 1)$ be the probability that agent i escaped the room. Then, logistic regression can be expressed as follows:

$$\log\left(\frac{p}{1-p}\right) = \beta_0(1,\ldots,1)^T + X\beta^T \tag{11}$$

where β_0 and $\beta = (\beta_1,\ldots,\beta_J)$ are the coefficients of a linear regression. The logistic regression method uses maximum likelihood optimization method to estimate the coefficients $\beta_0, \beta_1, \ldots, \beta_J$.

In the following sections, we describe how we selected the features to make sure the model fits the data well and avoid overfitting.

7.1 Model Selection

We started with the model (LogRegAll) that had all of the parameters mentioned in the previous section and used the R function "glm" (generalized linear models) to predict the coefficients using the data from Scenario 2 Run 0. The output is shown in Fig. 12. Then, we moved on to the model (LogRegSig) that only includes significant variables according to the Wald test p-value with a significance level $\alpha = 0.1$. The variables that were selected are the following: distance from the starting position of an agent to the exit, radius of an agent, agent body force, and proximity force between an agent and a wall (wall_a). The output is shown in Fig. 13.

To compare these two models, we used Akaike information criterion (AIC) [3], which is an estimator of the relative quality of statistical models. The AIC value is a difference between 2 times the number of features used in the model and 2 times the log-likelihood of this model. The smaller the value of AIC, the better the model in terms of balance between number of features used and probability of fitting the data. It is shown in [39] that Bayesian analog of AIC criteria, which is called BIC (Bayesian information criterion) is better since the probability that it finds the true model is 100%; however, there is no guarantee that the true model is one of the models to be tested.

It turned out that the model that included only significant variables LogRegSig has a smaller AIC value of 65.5, compared to 132.7 for the full model LogRegAll. Furthermore, we performed backward and forward selections to find the model that has the smallest AIC value. Both of these selection methods add or eliminate variables one by one and keep track of AIC values, then pick the model that has the smallest AIC. The smallest AIC value that was found by both forward and backward selections is 65.316. It corresponds to a model LogRegAIC which includes distance from the starting position of an agent to the exit, radius, agent body force, and wall_a variables that are the same as in LogRegSig model and, in addition, the

```
glm(formula = StopPosition ~ dist + R + A + PST + ARI + QR +
    BF + ABF + SFF + AB + AA + WB + WA + MS, family = binomial,
    data = data1)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.95831  -0.30630  -0.05766   0.27623   2.83440

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.498e+01  6.320e+00   2.370   0.0178 *
dist        -3.634e-02  1.989e-02  -1.826   0.0678 .
R           -2.788e+01  6.046e+00  -4.612 3.99e-06 ***
A           -2.185e+00  3.353e+00  -0.652   0.5145
PST          3.942e+00  5.029e+00   0.784   0.4331
ARI          4.105e+00  4.970e+00   0.826   0.4088
QR          -7.327e-01  5.504e-01  -1.331   0.1831
BF          -1.109e-04  9.687e-04  -0.114   0.9089
ABF          1.752e-03  1.036e-03   1.691   0.0909 .
SFF          5.729e-04  5.167e-04   1.109   0.2676
AB          -2.025e+01  2.053e+01  -0.986   0.3240
AA           4.697e-02  5.914e-02   0.794   0.4270
WB           1.193e+00  1.764e+01   0.068   0.9461
WA           1.119e-01  6.211e-02   1.801   0.0717 .
MS           8.228e-02  6.536e-01   0.126   0.8998
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 130.684  on 99  degrees of freedom
Residual deviance:  47.145  on 85  degrees of freedom
AIC: 77.145

Number of Fisher Scoring iterations: 7
```

Fig. 12 Logistic regression output for the model with all the parameters. In this figure, dist, R, A, PST, ARI, QR, BF, ABF, SFF, AB, AA, WB, WA, and MS mean Euclidean distance from the starting position of an agent to the exit, radius, acceleration, personal space threshold, agent repulsion importance, query radius, body force, agent body force, sliding friction force, agent_b, agent_a, wall_b, wall_b, and maximum speed, respectively

factor of frictional force. We decided to use LogRegSig model since AIC criteria selected the same features. On the other hand, we decided not to include factor of frictional force for now since it was not picked up by Wald test as having a significant p-value under the level of significance of 0.1 and the reason could be that this simulation study only consisted of 100 agents and thus the friction force did not have a strong effect when agents were moving. In a situation with a higher density of agents, we recommend including factor of frictional force as it is an important factor when we think about the process of evacuation with a lot of agents moving around.

```
Call:
glm(formula = StopPosition ~ dist + R + ABF + WA, family = binomial,
    data = data1)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.73658  -0.35569  -0.05643   0.38558   2.65521

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.381e+01  3.359e+00   4.112 3.93e-05 ***
dist        -2.089e-02  1.473e-02  -1.419    0.156
R           -2.447e+01  5.046e+00  -4.850 1.24e-06 ***
ABF          1.144e-03  7.661e-04   1.493    0.135
WA           8.018e-02  4.953e-02   1.619    0.105
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 130.684  on 99  degrees of freedom
Residual deviance:  55.316  on 95  degrees of freedom
AIC: 65.316
```

Fig. 13 Logistic regression output for the model that only included parameters with significant p-values, under level of significance of $\alpha = 0.1$. In this figure, dist, R, ABF, and WA mean Euclidean distance from the starting position of an agent to the exit, radius, acceleration, personal space threshold, and wall_a, respectively

7.2 Model Validation

We validated our model by predicting agents' exit probabilities for Runs 0 through 20 for Scenario 2 and then computing three measures of performance: sensitivity, specificity, and classification rate. They are defined as follows. Let Y_i be the true value for observation i, and $\hat{Y}_i$ be the predicted exit probability.

1. Sensitivity: $P(\hat{Y}_i = 1 | Y_i = 1)$—the proportion of 1's that are correctly identified as so.
2. Specificity: $P(\hat{Y}_i = 0 | Y_i = 0)$—the proportion of 0's that are correctly identified as so
3. Classification rate: $P(Y_i = \hat{Y}_i)$—the proportion of predictions that were correct.

The following Table 2 includes values of the original escape rates for each of the runs of Scenario 2 along with sensitivity, specificity, and classification rate.

The classification rates for all the runs are high. Similarly, the sensitivity rate is high, but the specificity rate is not as high. One of the reasons could be that the original escape probability is so high that it is harder to pick up "0" as there are not many. Indeed, we can see that for Run 0 the original escape rate is 36%, which is more balanced than other runs and thus the sensitivity and specificity rates are both high and relatively the same with specificity slightly higher since there are slightly more 0s in this run. One of the ways to handle unbalanced data is to randomly

Table 2 This table presents values of original escape rate, sensitivity rate, specificity rate, and overall classification rate

Run	Original escape rate	Sensitivity	Specificity	Classification rate
0	0.36	0.86	0.92	0.90
1	0.95	0.98	0.20	0.95
2	0.98	0.96	0.50	0.95
3	0.95	0.98	0.60	0.97
4	0.94	0.98	0.50	0.95
5	0.94	0.98	0.83	0.98
6	0.93	0.95	0.42	0.91
7	0.93	0.96	0.57	0.93
8	0.94	0.97	0.50	0.94
9	0.92	0.99	0.87	0.98
10	0.93	0.93	0.57	0.91
11	0.93	0.98	0.86	0.97
12	0.91	0.98	0.89	0.97
13	0.89	0.92	0.81	0.91
14	0.94	0.95	0.67	0.94
15	0.97	1	1	1
16	0.95	0.99	0.80	0.98
17	0.94	0.98	0.67	0.96
18	0.94	0.99	0.83	0.98
19	0.95	0.97	0.60	0.95
20	0.93	0.98	0.71	0.96

sample from the class of "1s" to make the rate closer to 50% however that would introduce selection bias. As a future research direction, we are planning to use an alternative estimation method that is proposed by King and Zeng [22] to reduce the bias. It is similar to penalized likelihood, which is a general approach for reducing small-sample bias in maximum likelihood estimation, an estimation method that is used in logistic regression to estimate the coefficients.

8 Methods for Estimating Exit Times

In this section, we explore machine learning techniques to estimate the exit times of N agents, using a feed-forward neural network for supervised multi-output regression. Based upon a number of experiments, we have settled on a "sliding window" approach for feature generation, whereby a particular agent is represented by features derived *across a number of time slices*, rather than restricting ourselves to a more naive agent representation involving information at only a single time step. Such a representation promised to provide a richer representation for each agent and thereby provide for more robust predictions.

In particular, our methodology follows a three-stage process. First, we perform feature engineering on the original data by using a "sliding window technique." Second, we take the input features and perform various types of *dimensionality reduction*. Herein, we study the differences between *linear* dimensionality reduction using principal component analysis (PCA) and using an auto-encoder to do nonlinear dimensionality reduction. In both cases, after the new lower-dimensional features are produced, a feed-forward neural network is used to make predictions of the agent exit times. Our proposed methodology promises a number of advantages. First, since the dimensionality reduction is performed without access to the measured exit times of the agents, at least that part of our procedure is safe from overfitting. Second, as the problem of interest is quite complex, there is likely a nonlinear relationship between our measured features (e.g., the initial position of the agents) and their final exit time. Accordingly, we hope to reduce bias by using nonlinear techniques in both our dimensionality reduction and in our final predictions. Note that the nonlinearity of the auto-encoder comes from the activation function. A purely linear activation function results in something, close to, if not PCA [7]. On the other hand, our experiments with auto-encoders gave unsatisfactory results using a piecewise linear function such as ReLu. Linear discriminant analysis (LDA) can be used but as a dimensionality reduction method as it is supervised, unlike PCA. In this paper, we are comparing two unsupervised dimensionality reduction methods (linear versus nonlinear). In this section and the following, we focus on performing feature engineering on the original features in order to improve the exit time predictions (and not on investigating the relevant features influencing the output).

Accordingly, the outline of our main experiments is as follows:

1. Perform feature engineering to generate a new data set using a subset of the original data.
2. Perform dimensionality reduction using either PCA (for a linear projection) or a 3-layer auto-encoder (for a nonlinear projection).
3. (a) If using PCA, then use the projected features as the predictors for our supervised learning.
   (b) If using an auto-encoder, then use the hidden layer as the predictors for our supervised learning.
4. Provide a training sample of our projected data to a 3-layer feed-forward neural network (from Sect. 9.1) to make predictions of the agents' exit times.

In Sect. 8.1, we describe the featuring engineering step consisting on a "sliding window" approach. While space does not allow for a fulsome treatment of dimensionality reduction methods, to make the prose self-contained we provide a brief discussion of the fundamentals of PCA and auto-encoders in Sects. 8.2 and 8.3. In addition, in Sect. 8.4 we introduce the fundamentals of *feed-forward* neural networks.

8.1 Feature Engineering: Sliding Window Approach

Typical use of feed-forward networks which employ a sliding window approach is in market predictions, and meteorological and network traffic forecasting [6, 11, 12, 14]. In the context of computer vision, a "sliding window" is rectangular region of fixed width and height that "slides" across an image [5]. In our context, the sliding window would be moving across slices of time.

We first describe the "sliding window" approach applied to a fixed agent and fixed run. Consider the set of features F_0, F_1, F_2, F_3, and F_4 associated to the first time steps $t_0, \ldots, t_4$. Each set of features F_m has $d = 21$ features, since the original data has 23 features and we eliminated the agent id and the time step. We have chosen scenario 19 that has 21 runs (see Appendix for a detailed description of building configurations for the scenarios) for all experiments. Let s_w be the size of the window ($1 \leq s_w \leq 5$) and n_w the number of windows. The first window is the first s_w set of features, i.e., $\{F_0, \ldots, F_{s_w-1}\}$, the second window is $\{F_1, \ldots, F_{s_w}\}$, and we continue the process until we select the last window of features $\{F_{n_w-1}, \ldots, F_{n_w+s_w-2}\}$, provided $n_w + s_w - 2$ is not greater than the total of time steps. We concatenate all the set of features of each window to generate a point of the new data set. For an illustration of the latter process in the case $n_w = 4$ and $s_w = 2$ for a fixed agent and run, see Fig. 14. Note that, in this case, the windows overlap, but we also can have nonoverlapping windows.

Since the stopping time of any agent should depend on the position of all the other agents, we concatenate the data for all $N = 100$ agents. Now, fix a run and let $F_m^{(i)}$ be the feature set for the i-th agent corresponding to the time step t_m. We did not want to add an index j to denote run j in order to simplify the exposition. Once the sliding technique has been performed for each agent , we concatenate all the set of features $\{F_m^{(i)}\}$ with $m = 0, \ldots, s_w - 1$ and $i = 1, \ldots, 100$. We end up with an array R_j, where $j = 1, \ldots, 21$ denotes the run label. Finally, we put all the arrays R_j's together and this represents the new data set (see Fig. 15 for a detailed description of the process with all agents and runs). This is the input for the given dimensionality reduction step. Moreover, each data point has dimension $p = d \times s_w \times N$, and the number of points is $n_w \times$ Total number of runs. In particular, in Fig. 15, $p = 21 \times 2 \times 100 = 4,200$ and $4 \times 21 = 84$ data points have been generated.

8.2 Principal Component Analysis (PCA)

PCA is one of the most popular unsupervised learning techniques and it performs linear dimensionality reduction that preserves as much of the variance in the data as possible after embedding the data into a linear subspace of lower dimension. Herein, we follow the exposition in [16] very closely, and the interested reader can look there for additional details.

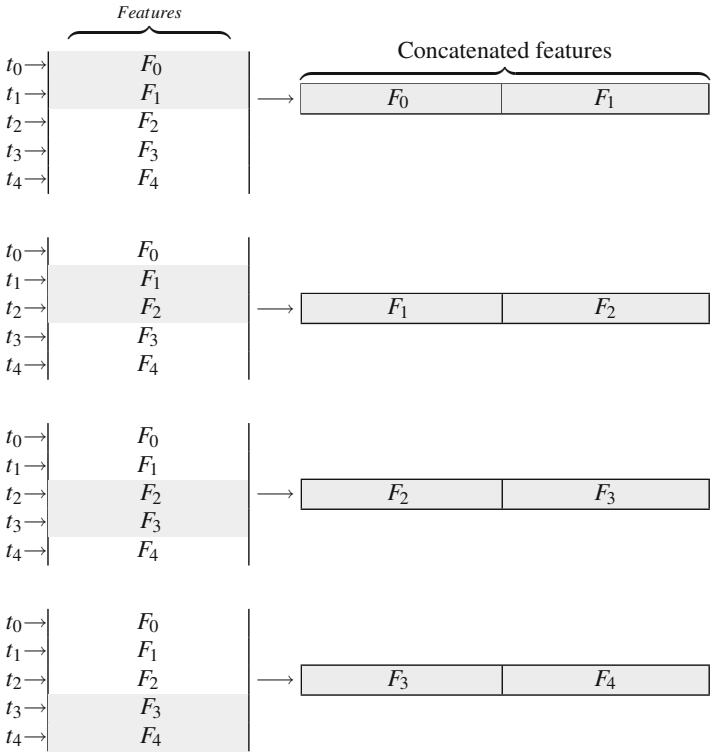

Fig. 14 Illustration of the "sliding window" technique for a fixed agent and run. This technique helps us generate a bigger data set of higher (or equal) dimension from a subset of the original data (data points corresponding to time steps t_0, t_1, t_2, t_3, and t_4). The number of generated new data points is proportional to the number of windows, and the dimension is proportional to the size of the window. In this case, we have 4 windows of size 2 each. We consider the following subsets of time steps $\{t_0, t_1\}$, $\{t_1, t_2\}$, $\{t_2, t_3\}$, and $\{t_3, t_4\}$

Suppose we have n observations of the p features $X_1,\ X_2,\ \ldots, X_p$. If we assume that we have zero-mean data, then each of the basis vectors of the low-dimensional subspace found by PCA are linear combinations of the original p features. The first principal component Z_1 is the linear combination of the p features:

$$Z_1 = \phi_{11} X_1 + \phi_{21} X_2 + \cdots + \phi_{p1} X_p$$

with largest variance, where $\phi_{11}, \phi_{21}, \ldots, \phi_{p1}$ are scalars such that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$, and $\phi_1 = (\phi_{11}, \phi_{21}, \ldots, \phi_{p1})^T$. We then look for the linear combinations of sample feature values $x_{i1},\ x_{i2}, \ldots, x_{ip}$

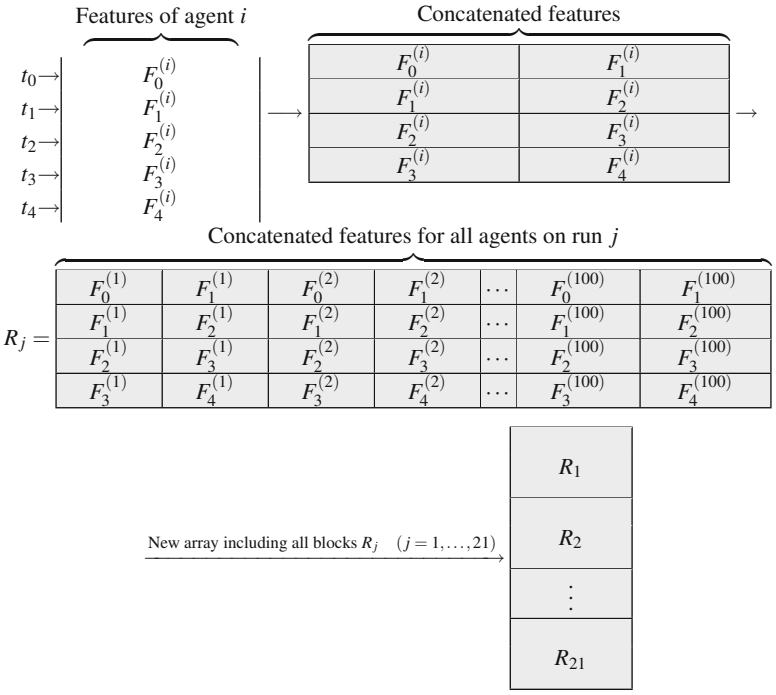

Fig. 15 Considering the features for agent i, we obtain a new array using the "sliding window" technique. We then concatenate the features for all 100 agents to obtain a new array for run j, R_j. Finally, we put all 21 runs together to form the new generated data set to feed PCA or auto-encoder. Note that in this case, we have 4 windows of size 2, so the dimension of the new data is $p = d \times s_w \times N = 21 \times 2 \times 100 = 4,200$. The number of points is $n_w \times$ number of runs $= 4 \times 21 = 84$

$$z_{i1} = \phi_{11} x_{i1} + \phi_{21} x_{i2} + \cdots + \phi_{1p} x_{ip}$$

that has largest sample variance subject to the constraint $\|\phi_1\|_2 = 1$. The values $z_{11}, z_{21}, \ldots z_{n1}$ are the scores of the first principal component Z_1.

Let X be an $n \times p$ data set. The first principal component loading vector ϕ_1 solves the optimization problem:

$$\underset{\phi_{11}, \phi_{21}, \ldots, \phi_{p1}}{maximize} \frac{1}{n} \sum_{i=1}^{n} z_{i1}^2 \text{ subject to } \|\phi_1\|_2 = 1, \tag{12}$$

In other words, the vector ϕ_1 defines the direction in the feature space along which the data has maximum variability. Projecting the data points onto this direction gives the principal components score values $z_{11}, z_{21}, \ldots, z_{n1}$. The optimization problem given by (12) can be solved via an eigen-decomposition.

In order to find the second principal component Z_2, we find the linear combination of maximum variance among all linear combinations uncorrelated to Z_1 (note that the process is equivalent to forcing ϕ_2 to be orthogonal to ϕ_1). We are solving now an optimization problem similar to (12) by replacing ϕ_1 by ϕ_2 (and replacing scores z_{i1} by z_{i2} with $i = 1, \ldots, n$).

Let $cov(X)$ be the sample covariance matrix of X. The principal components $\phi_1, \phi_2, \ldots, \phi_d$ are the ordered sequence of eigenvectors of $cov(X)$, and the variances of the components are the eigenvalues. PCA solves the eigen-problem:

$$cov(X)M = \lambda M, \tag{13}$$

where M is the matrix with columns ϕ_i, $i = 1, \ldots, d$. The eigen-problem (13) is solved by the d eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_d$. The low-dimensional data representation is obtained by mapping the data via M, that is:

$$Z = XM.$$

In our first set of experiments, we therefore feed the neural network with Z.

8.3 Auto-Encoders

Deep auto-encoders are feed-forward neural networks with an odd number of hidden layers and shared weights between the left and right layers. The input data X (input layer) and the output data $\hat{X}$ (output layer) have $d^{(0)}$ nodes (for a more detailed description on neural networks, see Sect. 8.4). More precisely, auto-encoders learn a nonlinear map from the input to itself through a pair of encoding and decoding phases [40, 41]:

$$\hat{X} = D(E(X)), \tag{14}$$

where E maps the input layer X to the "most" hidden layer (*encodes* the input data) in a nonlinear fashion, D is a nonlinear map from the "most" hidden layer to the output layer (decodes the "most" hidden layer), and $\hat{X}$ is the recovered version of the input data. An auto-encoder therefore solves the optimization problem:

$$\underset{E,\, D}{\operatorname{argmin}} \|X - D(E(X))\|_2^2, \tag{15}$$

We are motivated to include deep auto-encoders (or multilayer auto-encoders) in our exploratory analysis in crowd flow data, since they have demonstrated to be effective for discovering nonlinear features across problem domains. We first describe a 1-layer auto-encoder to prepare the reader for a more complex auto-encoder (more layers). Then, we describe a *3-layer auto-encoder*.

8.3.1 1-Layer Auto-Encoder

A 1-layer auto-encoder consists of three layers: the input layer $X \in \mathbb{R}^{d^{(0)}}$, a single-hidden layer $Z \in \mathbb{R}^{d^{(1)}}$, and an output layer $\hat{X} \in \mathbb{R}^{d^{(0)}}$. Note that for auto-encoders the output dimension is the same as the input dimension. We aim to find maps $E = f$ and $D = f^+$ such that $f(X) = Z$ and $f^+(Z) = \hat{X}$ ($d^{(0)} > d^{(1)}$), which solve the optimization problem:

$$\underset{W^{(1)},b_1,c_1}{\operatorname{argmin}} \|X - f^+(f(X))\|_2^2 = \underset{W^{(1)},b_1,c_1}{\operatorname{argmin}} \|X - \hat{X}\|_2^2, \tag{16}$$

where $W^{(1)}$ are the weights, and b_1 and c_1 are coming from the bias term. They determine the nonlinear maps f (encoder) and f^+ (decoder). Note that the nonlinearity comes from the introduction of some activation function θ :

$$f(X) = \theta\left(W^{(1)}X + b_1\right), \quad f^+(Z) = \theta\left((W^{(1)})^T Z + c_1\right). \tag{17}$$

The nonlinearity of the auto-encoder comes from the activation function θ. We have chosen the sigmoid function $\theta(v) = \dfrac{1}{1+e^{-v}}$ as activation function for the auto-encoders used in this paper. The other common activation function used for auto-encoders is $\mathrm{ReLu}(v) = \max(0, v)$, but we obtained unsatisfactory results when compared to the sigmoid function.

As suggested in the introduction of this section, the auto-encoder $h_{\mathbf{w}}(X) = f^+(f(X))$ "tries to learn" the identity function:

$$\hat{X} = h_{\mathbf{w}}(X) \approx X. \tag{18}$$

Figure 16 shows a diagram of a 1-layer auto-encoder.

8.3.2 3-Layer Auto-Encoder

We now describe an auto-encoder with three inner layers. We aim to find functions f and g which are solutions to the corresponding optimization problem (19), with first hidden layer (leftmost hidden layer) $S_l \in \mathbb{R}^{d^{(1)}}$, "deepest hidden layer" $Z \in \mathbb{R}^{d^{(3)}}$, and "rightmost inner layer" $S_r \in \mathbb{R}^{d^{(3)}}$.

$$\underset{W^{(1)},W^{(2)},b_1,c_1,b_2,c_2}{\operatorname{argmin}} \|X - f^+(g^+(g(f(X))))\|_2^2 = \underset{W^{(1)},W^{(2)},b_1,c_1,b_2,c_2}{\operatorname{argmin}} \|X - \hat{X}\|_2^2, \tag{19}$$

where

$$f(X) = \theta\left(W^{(1)}X + b_1\right) = S_l, \quad g(S_l) = \theta\left((W^{(2)})^T S_l + c_2\right) = Z, \tag{20}$$

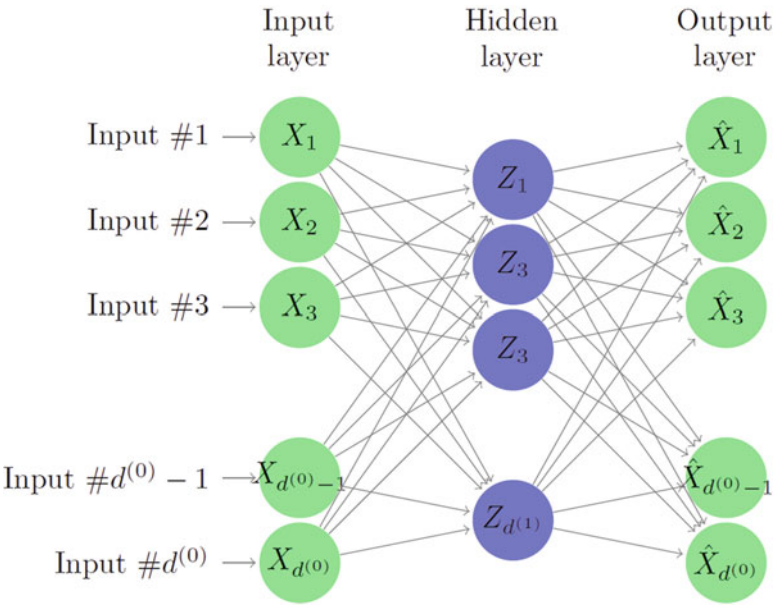

Fig. 16 Single-layer auto-encoder diagram. Input X, inner Z and $\hat{X}$ have dimensions $d^{(0)}$, $d^{(1)}$ and $d^{(2)} = d^{(0)}$ (same as input layer), respectively

and

$$g^+(Z) = \theta(W^{(2)}Z + b_2) = S_r, \quad f^+(S_r) = \theta\left((W^{(1)})^T S_r + c_1\right) = \hat{X}, \tag{21}$$

where θ is the activation function. Notice that we are increasing the complexity of the auto-encoder architecture by adding one more layer to a 1-layer auto-encoder (described in Sect. 8.3.1). We then have that the nonlinear map composition $E = g \circ f$ is the encoder, and the map $D = f^+ \circ g^+$ is the decoder for this particular deep auto-encoder. For an illustration of the auto-encoder just described, see Fig. 17.

In the next section, we perform dimensionality reduction by applying the encoder E to the input layer X. We then feed the neural network with $Z = E(X)$.

8.4 *Neural Network Fundamentals*

Neural networks provide a practical method for learning real-valued, discrete-valued, and vector-valued functions from examples. Leaning to interpret real-world sensor data, and learning to recognize faces and handwritten characters are among the problems where artificial neural networks perform as an effective learning method [29]. In this paper, we consider a multi-output regression. We have followed the exposition as well as some of the notation in [2] and [16] to some extent.

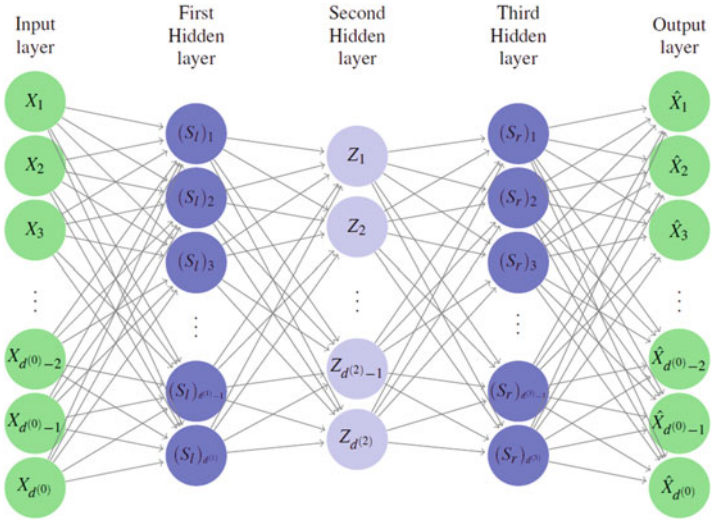

Fig. 17 Three-layer auto-encoder diagram. The input layer has dimension $d^{(0)}$, the three inner layers S_l, Z, and S_r have dimensions $d^{(1)}$, $d^{(2)}$, and $d^{(3)}$, respectively. The dimension of the outer layer $\hat{X}$ has dimension $d^{(0)}$ since this is an auto-encoder

Figure 18 represents a neural network diagram, the usual way of representing neural networks. The index l refers to layer l, where $l = 0, 1, \ldots, L$. Here, the input layer is the leftmost layer. The additional middle layers are called *hidden layers*, and there are $L - 2$ *inner layers*. In the case of Fig. 18, $L = 3$ so we just have one hidden layer, but in general we can have as many hidden layers as we want. We can also choose how many *nodes* (or units) are on each layer, which is called the layer dimension. We denote by $d^{(l)}$ the dimension of layer l. For example, $d^{(0)}$ denotes the dimension (number of nodes or units) of the input layer. The nodes or unit on each layer are represented by the circles as seen in Fig. 18. We focus on the class of *feed-forward* neural networks, which means that there are no backward pointing arrows and no jumps to other layers.

Once we fix the number of layers and nodes on each layer, we choose an algorithm that learns the weights on each link (arrow) by fitting the data. Every arrow represents a weight or connection between a node in a layer to another node in the next higher layer. We denote the complete input as a vector $X = \left(X_1, X_2, X_3, \ldots, X_{d^{(0)}}\right) \in \mathbb{R}^{d^{(0)}}$ (the original input vector is augmented with $X_0 = 1$).

We need to find optimal weights for the whole system. Every node has a transformation function θ. From layer $l-1$ to layer l, we have a weight matrix $W^{(l)}$ (weights in) of size $d^{(l-1)} \times d^{(l)}$, and the matrix $W^{(l+1)}$ (weights out) of dimension $d^{(l)} \times d^{(l+1)}$. We put all weight matrices together and represent them in a weight parameter $\mathbf{w} = \{W^{(1)}, W^{(2)}, \ldots, W^{(L)}\}$. The approximation to the target function is denoted by $h_{\mathbf{w}}(X)$ to indicate dependence on the weights $\mathbf{w}$.

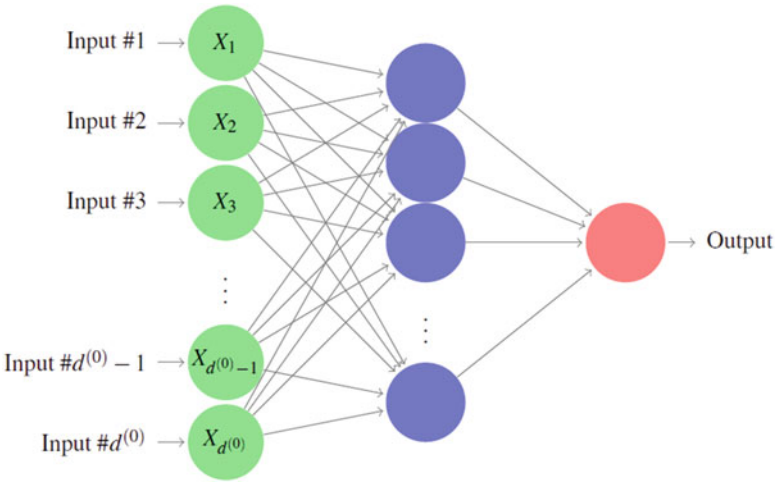

Fig. 18 Neural network with a single-hidden layer. The input layer has $d^{(0)}$ nodes (units), hidden layer has $d^{(1)}$ units, and output layer $d^{(2)}$ units

In our application, derived features S_m (hidden units) are created from linear combinations of the inputs X_i, and the targets T_k (stopping times in our application) are modeled as functions of linear combinations S_m. For our problem, the activation function θ is the sigmoid σ given by $\sigma(v) = \dfrac{1}{1+e^{-u}}$.

To find the weight in $\mathbf{w}$, it is common to use the *batch gradient descent* algorithm. The details of the algorithm are out of the scope of this paper (see [2] for a detailed explanation).

9 Estimation of Exit Times

In this section, we aim to estimate the exit time of $N = 100$ agents using a neural network with dimensionality reduction (PCA and auto-encoder). The input for the neural network has dimension $d^{(0)} = 20$ as the explained variance for PCA is greater than 95% when using 20 components, and the output has dimension $d^{(L)} = 100$ which are exit times for each agent.

Since this is a multi-output regression, we computed the R^2-scores or coefficient of determination:

$$R^2 = 1 - \frac{\|h(X) - T\|_2^2}{\|T - \bar{T}\|_2^2} \tag{22}$$

where $\bar{T} = \frac{1}{N}\sum_{i=1}^{N} T_i$.

An R^2 score near 1 means that the model is predicting the data (stopping time) very well. A score close to zero means that the model is predicting the *mean* of the stopping times and the score can be arbitrarily negative indicating bad predictions by the model.

In all experiments, we split the initial input data. We train the learning algorithm with 80% of the data and test it with the remaining 20% of the data (we do appropriate normalization of the data, z-scores). Also, we have considered a multi-linear regression for the learning algorithm to produce stopping times for $N = 100$ agents at once.

9.1 Experiments: PCA vs Auto-Encoder with Neural Networks

We present two main sets of experiments. The first set involves feature engineering using PCA to produce new features to feed a forward neural network. For the second set, we train a 3-layer auto-encoder instead of using PCA.

In all experiments, the neural network architecture consists of an input layer made of 20 inputs $(Z_1, \ldots, Z_{20})$ obtained after dimensionality reduction, two hidden layers (first hidden layer has 50 units, and second hidden layer has 70 units). The output layer consists of 100 units representing exit time for each of the 100 agents (see Fig. 19.)

We had little data for training, so we use a "sliding window" technique (see Fig. 14 for illustration) in order to produce more data points (each point has higher dimension depending on the window size) and improve the performance of the neural network. Overlapping windows and non-overlapping windows are considered. Note that the input data that gives the least number of data points corresponds to window size 1, which includes the original data together with all runs for a fixed scenario at one single time step. The windows don't overlap.

We have used TensorFlow (an open source software library for numerical computation using data flow graphs, see [1]) to build the auto-encoder. The rest of the scripts are in Python using Sci-kit Learn [33] and Pandas [28] libraries.

9.1.1 Experiments with PCA

Recall that our main steps for the proposed algorithm are as follows:

1. Generate new data X from the original data by using the sliding window approach (as described in Sec. 8.1).
2. Perform PCA on the new input data X to obtain a new input layer Z.
3. Feed the original 2-layer neural network with $Z = XM \in \mathbb{R}^{20}$.

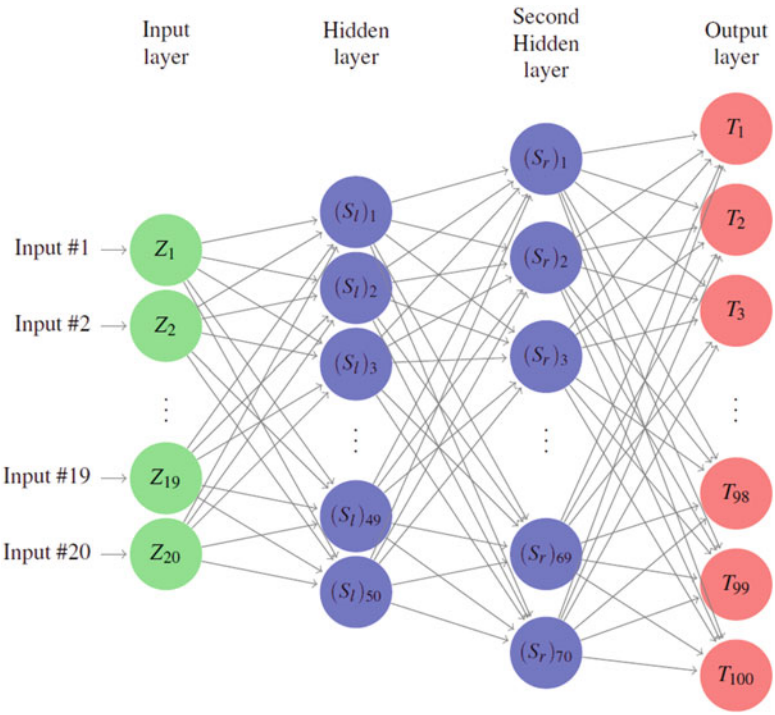

Fig. 19 Neural network with input layer obtained after dimensionality reduction (20 inputs), two hidden layers, and output layer with estimated exit times $T_1, T_2, \ldots, T_{100}$

We reduce the dimension of the space performing PCA. The cumulative variance is computed to get an estimate for the number of components. We end up choosing 20 components. The input layer now consists of $Z \in \mathbb{R}^{d^{(0)}}$ with $d^{(0)} = 20$. The results for this particular experiments were not satisfactory as we observe that in Table 4 (row 1), the score is -7.57312.

After augmenting the dimension of the input layer, we perform PCA to reduce the dimension to 20, and then we train a neural network with two hidden layers ($L = 4$). The input layer has dimension $d^{(0)} = 20$ and the output, $d^{(3)} = 100$, so the output layer contains T the stopping times for each agent. The hidden layers have dimensions $d^{(1)} = 50$ and $d^{(2)} = 70$.

We summarize scores that we have obtained with 1, 2, and 3 windows with different sizes when using PCA (see Tables 3–6).

Note that we have made the above experiments with scenarios 19 and 8 and the scores are very similar (see Tables 3–6).

Table 3 With PCA with overlapping windows for scenario 19

Window size	Number of windows	R^2-score
2	1	−8.45903
	2	−0.34136
	3	−0.19203
	4	0.99402
3	1	−7.702292
	2	−2.15774
	3	−0.54476
	4	0.99563

Table 4 With PCA with **no** overlapping windows for scenario 19

Window size	Number of windows	R^2-score
1	1	−7.57312
	2	−0.28829
	3	0.99599
2	1	−8.04351
	2	−1.72193
	3	0.99815
3	1	−8.02079
	2	−1.74902
	3	0.99338

Table 5 With PCA with overlapping windows for scenario 8

Window size	Number of windows	R^2-score
2	1	−7.32216
	2	−1.60795
	3	0.99526
3	1	−8.62982
	2	−0.39852
	3	0.99938

Table 6 With PCA with **no** overlapping windows for scenario 8

Window size	Number of windows	R^2-score
1	1	−6.81011
	2	−3.89427
	3	0.99990
2	1	−7.60565
	2	−1.35560
	3	0.99917
3	1	−6.50695
	2	−0.14562
	3	0.99872

9.1.2 Experiments with Auto-Encoder

In these experiments, we consider the same features and layers as in the experiments from Sect. 9.1.1. But now, we reduce the dimension of the data using the encoder E from the corresponding auto-encoder (see Sect. 8.3.2 instead of PCA). We construct the auto-encoder with two inner layers of dimensions 500 and 20 (in that order). Recall our main steps are as follows:

1. Generate new data X from the original data by using the sliding window approach (as described in Sec. 8.1).
2. Construct a 3-layer auto-encoder.
3. Extract the encoder E and apply it to the initial data to reduce the dimension to 20.
4. Feed the original 2-layer neural network with $Z = E(X) \in \mathbb{R}^{20}$.

9.2 Exit Time Estimation Results Summary

We can see a big improvement whenever we use the sliding window technique as seen in Tables 3, 4, and 7 when the window size is equal or greater than 2. Also, when using the proposed 3-layer auto-encoder we observe a small improvement on the performance of our learning algorithm with respect to PCA on scenario 19 (compare Tables 5 and 7). Indeed, observe that we have positive scores for window size 3 when the number of windows is 2 and up when using an auto-encoder (Table 7) and when using PCA with overlapping windows in Table 5 scores are all negative except for the last one (0.99563). However, in the aforementioned tables, the score learning algorithm involving PCA is better than the one for auto-encoder when using window size 3 and 4 overlapping windows. Notice that in all PCA experiments there is an important jump in accuracy for PCA when we go from a single window to 2 windows. For example, in Table 3, the score is $-845,903$

Table 7 R^2 for encoder (from 2-layer auto-encoder) with overlapping windows for scenario 19

Window size	Number of windows	R^2-score
2	1	−0.54033
	2	−0.03344
	3	−0.19686
	4	0.04300
3	1	−0.86243
	2	0.05440
	3	0.23429
	4	0.20910

using a single window and -0.34136 using 2 windows for a window of size 2. On the other hand, increasing the window size does not improve accuracy and in some cases, the accuracy decreases. For example, Table 3 window size 2 and number of windows 2, the score is -0.34136 and window size 3 and number of windows size 2 results in -2.15774

We emphasize to the reader that there are two large jumps in accuracy seen in experiments involving PCA. We attribute these large jumps to the fact that studying the agent for longer time provides substantially more information that leads to more accurate predictions. As seen on the experiment, increasing the number of windows increases the performance of the learning algorithms when using both dimensionality reduction methods. PCA resulted in more accurate predictions for 3 and 4 number of windows (with size greater than or equal to 2), and auto-encoders exhibited better scores for 1 and 2 number of windows. Average scores are better for auto-encoders. The nonlinear properties of auto-encoders allow for more accurate predictions from a small number of features. However, we don't have enough instances to train all parameters of the auto-encoder.

We can change the architecture of the neural network (augmenting the number of hidden layers or dimensionality of each layer) to see if the new encoder performs better than the architecture involving PCA.

10 Summary and Future Research Directions

Simulated crowd flow exit data generated by the SteerSuite platform [35] was analyzed by a 10-person mathematical sciences team consisting of experts with a broad range of theoretical and applied mathematical and statistical expertise. The data set consisted of simulated trajectories for 100 randomly placed agents in 19 different obstacle configurations for a one-story building. The (unknown) algorithm steered the agents to exit the building using the parameters of the agents and the positions of the other agents and the building walls and obstacles.

The trajectory data was first visualized using k-means clustering and then visualized by coloring the first two Principal Components of the trajectories by their starting time and radius. The trajectory visualization revealed that the trajectories were piecewise linear. Comparison of the PCA results and visualization with the trajectory visualization suggest that the starting point of a trajectory provided enough information that it could be used to represent an agent in some analyses. This idea was exploited in the Exit Time Estimation methods. Visualization also revealed that agents could be approximately clustered into two groups: peaceful agents (trajectories with large piecewise linear segments) and panicked agents who lost their direction in high-density situations and circled around the exit. An unsupervised algorithm based on the Normalized Cut Algorithm was developed and successfully demonstrated to do this clustering algorithmically.

Different exit convergence patterns were detected by an unsupervised analysis which represented the trajectories in each simulation run as a stochastic counting process and computed the sequence of distances to the static delta process at the exit. The average sequence of distances (a time series) was computed for each room configuration and visualized revealing 4 distinct exit convergence patterns. Some qualitative attribution analysis in terms of the agent parameters and properties of the room configurations was done. The qualitative attribution to a few agent parameters was reinforced by the quantitative logistic regression analysis. Since no quantitative room configuration statistics were provided with the original data set, a detailed manual analysis of differences between the room configurations was done. The measures in each stochastic process were represented in terms of the multi-scale product coefficients determined by a binary tree structure on the building space.

Logistic regression, using the 10 static parameters, was used to determine which parameters influence the probability of successfully exiting a room. The parameters distance to the exit, agent radius, agent body force, proximity force between an agent and a wall, and sliding friction force were selected. The selected model had very high classification rates for predicting exiting and nonexiting agents.

In the final analysis effort, machine learning techniques were used to estimate exit times of the agents, using a feed-forward neural network for supervised multi-output regression. A sliding window approach was used for feature generation and two types of dimensionality reduction were used: PCA and encoder–decoder. The results were evaluated using the R^2-score. Exit times were successfully predicted for certain combinations of the methods. The sliding window technique resulted in a big improvement in Exit Time prediction and use of the encoder resulted in a small improvement over PCA dimensionality reduction.

The analysis effort introduced the team to the very interesting and increasingly important research area of crowd dynamics. The research effort revealed a number of potential future research directions:

- Exploit shape features for the building configurations in the Exit Convergence Analysis, Logistic Regression Prediction of Probability of Escape, and Estimation of Exit Time.
- Determine relationships between encoder–decoders and product coefficient representations of measures
- Improve the robustness of the logistic regression model to predict probability of escape by using the trajectory data from all of the scenarios
- Analyze real-world tracking data from crowds at public events
- Perform more experiments with the “sliding window” technique:
  - Consider windows including early time steps and compare the exit time prediction with a set of windows using later time steps.
  - Consider a given agent A_j and a window that includes its features and features of neighbor agents. Then, consider agent A_j and agents located farther away. Which set of “windows” gives better predictions?

- Modify the architecture of the auto-encoder by adding more layers and/or changing the dimension of the inner layers. Compare the accuracy using this new preprocessing step with the one resulting from PCA.
- Modify the learning algorithm used to estimate exit times by using regression forest instead of multi-output linear regression.

Acknowledgements This research started at the *Women in Data Science and Mathematics Research Collaboration Workshop (WiSDM)*, July 17–21, 2017, at the *Institute for Computational and Experimental Research in Mathematics (ICERM)*. The workshop was partially supported by grant number NSF-HRD 1500481-AWM ADVANCE and co-sponsored by the Brown's Data Science Initiative. Subsequently, the team of collaborators expanded to include Boris Iskra, F. Patricia Medina, and Randy Paffenroth. We gratefully acknowledge their interest in and contributions to the research.

Additional support for some participant travel was provided by DIMACS in association with its Special Focus on Information Sharing and Dynamic Data Analysis. Linda Ness worked on this project during a visit to DIMACS, partially supported by the National Science Foundation under grant number CCF-1445755 and by DARPA SocialSim-W911NF-17-C-0098. Her work has also been funded in part by DARPA SocialSim-W911NF-17-C-0098. F. Patricia Medina received partial travel funding from Worcester Polytechnic Institute, Mathematical Science Department. This work was partially supported by a grant from the Simons Foundation (355824, MO).

Appendix: Building Configuration Descriptions

The positions and parameters in the agent tracking data were influenced by the building configurations. The images of all of the building configurations were provided (see Fig. 20), but a list of distinguishing features was not included. We quantitatively summarized the locations and orientations of obstacles and exits in the 19 building configuration images. The one-story building had 3 rooms on the north side of the building, 2 rooms on the east side of the building, and 2 rooms on the south side of the building, and a large common room in the remaining space. The placement and orientation of obstacles (3 bar objects, 4 rectangular objects, and 2 square objects) varied among the configurations. The location of the exit also varied. Below is a list of the features common to each building configuration. The results are shown in Table 8.

Following are a representation or a name for features common to each scenario:

1. Room 1N, 2N, and 3N Rooms on the North Side
2. Room 1E, 2E Rooms on the East Side
3. Room 1S, 2S Rooms on the South Side
4. Bar 1 → Rotating Lavender (L-Shaped) Bar
5. Bar 2 → Long Lavender Bar
6. Bar 3 → Short Lavender Bar

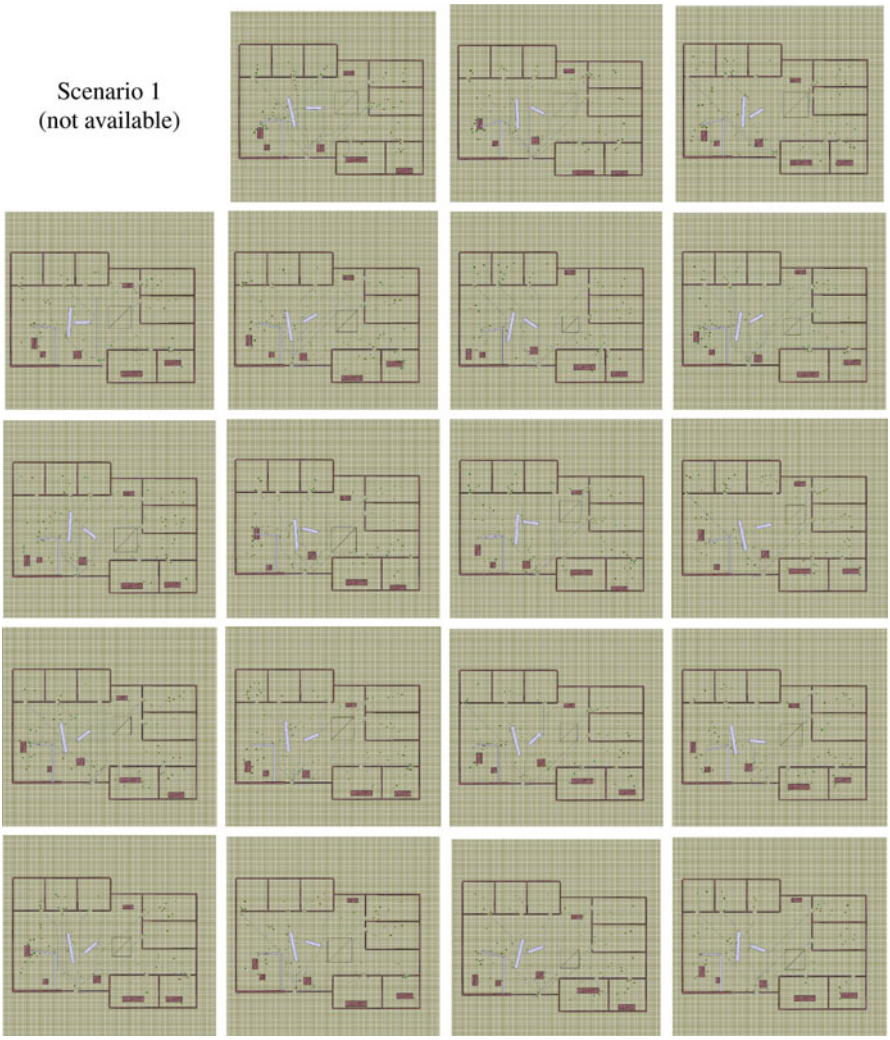

Fig. 20 Array of nineteen scenarios. Each row (from left to right): scenarios 2–4, 5–8, 9–12, 13–16, and 17–20

7. Rectangle 1S → Rectangle Box in Room 1S
8. Rectangle 2S → Rectangle Box in Room 2S
9. Rectangle 1NE → Rectangle in Between Room 3N and Room 1E
10. Square 1SE → Square Box near Exit and Room 1S
11. Square 1SW → Square Box near Exit and on West side
12. Rectangle 1SW → Rectangle box on West Side of the Common Room

Table 8 Positions and orientations of obstacles and exits in the 19-building scenario configurations

	Bar 1	Bar 2	Bar 3	Rec 2S	Rec 1S	Rec 1NE	Sq 1SE	Sq 1SW	Rec 1SW	EXIT
Sc 2	185°	10°	180°	(0, 0)	(0, 51)	(0, 41)	(0, 17)	(0, 22)	(0, 71)	(320, 500)
Sc 3	210°	5°	−30°	(0, 0)	(0, 0)	(0, 29)	(0, 60)	(0, 50)	(0, 135)	(320, 500)
Sc 4	195°	5°	20°	(0, 30)	(0, 30)	(0, 70)	(0, 35)	(0, 40)	(0, 65)	(414, 400)
Sc 5	175°	5°	5°	(0, 65)	(0, 2)	(0, 25)	(0, 25)	(0, 35)	(0, 75)	(414, 400)
Sc 6	150°	5°	30°	(0, 75)	(0, 0)	(0, 29)	(0, 11)	(0, 30)	(0, 80)	(250, 570)
Sc 7	195°	5°	135°	(0, 20)	(0, 50)	(0, 7)	(0, 47)	(0, 25)	(0, 15)	(355, 465)
Sc 8	185°	5°	15°	(0, 75)	(0, 30)	(0, 7)	(0, 11)	(0, 45)	(0, 95)	(290, 525)
Sc 9	180°	−5°	135°	(0, 20)	(0, 10)	(0, 55)	(0, 47)	(0, 55)	(0, 45)	(275, 545)
Sc 10	10°	5°	−15°	(0, 46)	(0, 15)	(0, 5)	(0, 14)	(0, 30)	(0, 66)	(320, 500)
Sc 11	180°	−5°	165°	(0, 0)	(0, 70)	(0, 60)	(0, 20)	(0, 20)	(0, 90)	(340, 480)
Sc 12	200°	5°	15°	(0, 75)	(0, 75)	(0, 10)	(0, 65)	(0, 10)	(0, 150)	(340, 480)
Sc 13	180°	5°	45°	(0, 0)	(0, 70)	(0, 45)	(0, 70)	(0, 50)	(0, 130)	(355, 460)
Sc 14	180°	5°	15°	(0, 5)	(0, 5)	(0, 75)	(0, 15)	(0, 20)	(0, 50)	(355, 460)
Sc 15	180°	25°	45°	(0, 20)	(0, 70)	(0, 70)	(0, 80)	(0, 40)	(0, 90)	(250, 570)
Sc 16	195°	5°	165°	(0, 75)	(0, 40)	(0, 70)	(0, 55)	(0, 40)	(0, 135)	(275, 540)
Sc 17	170°	15°	60°	(0, 20)	(0, 20)	(0, 40)	(0, 30)	(0, 20)	(0, 155)	(280, 545)
Sc 18	180°	15°	135°	(0, 40)	(0, 10)	(0, 15)	(0, 25)	(0, 40)	(0, 100)	(270, 560)
Sc 19	160°	−15°	5°	(0, 0)	(0, 80)	(0, 70)	(0, 60)	(0, 30)	(0, 35)	(340, 480)
Sc 20	180°	−2°	135°	(0, 40)	(0, 30)	(0, 5)	(0, 20)	(0, 30)	(0, 65)	(330,490)

The quantities in Table 8 characterize the variation in positions and orientations of the obstacles. The measurement methodology is described below for the objects listed in the columns of Table 8:

1. Bar 1 → angle in Degrees with Positive x-axis
2. Bar 2 → angle in Degree Angle with Negative y-axis
3. Bar 3 → angle in Degrees with Positive x-axis
4. Rec 2S = Rectangle Box in Room 1S → (0, Distance from South Wall of Room 2S)
5. Rec 1S = Rectangle Box in Room 2S → (0, Distance from South Wall of Room 1S)
6. Rec 1NE = Rectangle in Between Room 3N and Room 1E → (0, Distance from North Wall of Room 2S)
7. Sq 1SE = Square Box near Exit and Room 1S →(0, Distance from South Wall)
8. Sq 1SW = Square Box near Exit and on West side → (0, Distance from South Wall)
9. Rec 1SW = Rectangle box on West Side of the Common Room →(0, Distance from South Wall)
10. EXIT = EXIT from the whole building → (Distance from West Wall, Distance from East Wall)

References

1. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* (2015). tensorflow.org
2. Y.S. Abu-Mostafa, M. Magdon-Ismail, H.-T. Lin, *Learning From Data (e-Chapter)* (AML-Book, 2012)
3. H. Akaike, A new look at the statistical model identification. IEEE Trans. Autom. Control **19**, 716–723 (1974)
4. A. Alami, *Morocco Food Stampede Leaves 15 Dead and a Country Shaken* (The New York Times, 2017). https://www.nytimes.com/2017/11/19/world/africa/morocco-stampede.html. Accessed 1 Jan 2018
5. Y. Amit, P.F. Felzenszwalb, Object detection, in *Computer Vision, a Reference Guide* (2014), pp. 537–542
6. S. Bengio, F. Fessant, D. Collobert, A connectionist system for medium-term horizon time series prediction, in *Proceedings of the International Workshop on Application Neural Networks to Telecoms* (1995), pp. 308–315
7. Y. Bengio, A. Courville, P. Vincent, Representation learning: a review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. **35**, 1798–1828 (2013)
8. E. Bonabeau, Agent-based modeling: methods and techniques for simulating human systems. PNAS **99**, 7280–7287 (2002)
9. T. Bosse, R. Duell, Z.A. Memon, J. Treur, C.N. Van Der Wal, Multi-agent model for mutual absorption of emotions. ECMS **2009**, 212–218 (2009)
10. S. Curtis, A. Best, D. Manocha, Menge: a modular framework for simulating crowd movement. Collect. Dyn. **1**, 1–40 (2016)
11. G. Dorffner, Neural networks for time series processing. Neural Netw. World **6**, 447–468 (1996)
12. T. Edwards, D.S.W. Tansley, R.J. Frank, N. Davey, N.T. (Nortel Limited), Traffic trends analysis using neural networks, in *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications* (1997), pp. 157–164
13. R. Fefferman, C. Kenig, J. Pipher, The theory of weights and the Dirichlet problem for elliptical equations. Ann. Math. **134**, 65–124 (1991)
14. R.J. Frank, N. Davey, S.P. Hunt, Time series prediction and neural networks. J. Intell. Robot. Syst. **31**, 91–103 (2001)
15. R. Gladstone, *Death Toll From Hajj Stampede Reaches 2,411 in New Estimate* (The New York Times, 2015). https://www.nytimes.com/2015/12/11/world/middleeast/death-toll-from-hajj-stampede.html. Accessed 19 Dec 2017
16. T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, 2nd edn. (Springer, Berlin, 2009)
17. D. Helbing, P. Molnár, Social force model for pedestrian dynamics. Phys. Rev. E **51**, 4282–4286 (1995)
18. H. Hotelling, Analysis of a complex of statistical variables into principal components. J. Educ. Psychol. **24**, 417 (1933)
19. J.D. Hunter, Matplotlib: a 2D graphics environment. Comput. Sci. Eng. **9**, 90–95 (2007)
20. M. Kapadia, N. Pelechano, J. Allbeck, N. Badler, Virtual crowds: steps toward behavioral realism. Synth. Lect. Vis. Comput. **7**, 1–270 (2015)
21. M. Kapadia, S. Singh, W. Hewlett, P. Faloutsos, Egocentric affordance fields in pedestrian steering, in *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (ACM, New York, 2009), pp. 215–223

22. G. King, L. Zeng, Logistic regression in rare events data. Polit. Anal. **9**, 137–163 (2001)
23. A. Krontiris, K. Bekris, M. Kapadia, ACUMEN: activity-centric crowd monitoring using influence maps, in *Proceedings of the 29th International Conference on Computer Animation and Social Agents (CASA '16)* (ACM, New York), pp. 61–69
24. H. Kumar, *Stampede at Mumbai Railway Station Kills at Least 22* (The New York Times, 2017). https://www.nytimes.com/2017/09/29/world/asia/mumbai-railway-stampede-elphinstone.html. Accessed 1 Jan 2018
25. A. Lachapelle, M.-T. Wolfram, On a mean field game approach modeling congestion and aversion in pedestrian crowds. Transp. Res. B Methodol. **45**, 1572–1589 (2011)
26. U. Luxburg, A tutorial on spectral clustering. Stat. Comput. **17**, 395–416 (2007)
27. B. Maury, A. Roudneff-Chupin, F. Santambrogio, J. Venel, Handling congestion in crowd motion modeling. Netw. Heterog. Media **6**, 485–519 (2011)
28. W. McKinney, Data structures for statistical computing in python, in *Proceedings of the 9th Python in Science Conference*, ed. by S. van der Walt, J. Millman (2010), pp. 51–56
29. T.M. Mitchell, *Machine Learning* (McGraw-Hill, New York, 1997)
30. C.L. Mumford, *Computational Intelligence: Collaboration, Fusion and Emergence*, vol. 1 (Springer, Berlin, 2009)
31. L. Ness, Dyadic product formula representations of confidence measures and decision rules for dyadic data set samples, in *Proceedings of the 3rd Multidisciplinary International Social Networks Conference on Social Informatics 2016, Data Science 2016* (ACM, New York, 2016), pp. 35:1–35:8
32. K. Pearson, LIII. On lines and planes of closest fit to systems of points in space. Lon. Edinb. Dublin Philos. Mag. J. Sci. **2**, 559–572 (1901)
33. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
34. N. Pelechano, J.M. Allbeck, N.I. Badler, Virtual crowds: methods, simulation, and control. Synth. Lect. Comput. Graph. Animation **3**, 1–176 (2008)
35. S. Singh, M. Kapadia, P. Faloutsos, G. Reinman, An open framework for developing, evaluating, and sharing steering algorithms, in *Proceedings of the 2nd International Workshop on Motion in Games (MIG '09)* (Springer, Berlin, 2009), pp. 158–169
36. N. Sjarif, S. Shamsuddin, S. Hashim, Detection of abnormal behaviors in crowd scene: a review. Int. J. Adv. Soft Comput. Appl. 4, 1–33 (2012)
37. H. Swathi, G. Shivakumar, H. Mohana, Crowd behavior analysis: a survey, in *2017 International Conference on Recent Advances in Electronics and Communication Technology (ICRAECT)* (IEEE, Piscataway, 2017), pp. 169–178
38. L. Wang, M.B. Short, A.L. Bertozzi, Efficient numerical methods for multiscale crowd dynamics with emotional contagion. Math. Models Methods Appl. Sci. **27**, 205–230 (2017)
39. Y. Yang, Can the strengths of AIC and BIC be shared? A conflict between model identification and regression estimation. Biometrika **92**, 937–950 (2005)
40. D. Yu, L. Deng, Deep learning and its applications to signal and information processing. IEEE Signal Process. Mag. **28**, 145–54 (2011)
41. C. Zhou, R.C. Paffenroth, Anomaly detection with robust deep autoencoders, in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)* (ACM, New York, 2017), pp. 665–674