Predicting HIV Viral Body Load through Image Analysis

Ben Maldonado Darren Calovini Gabe Skidmore Junjiang Li Zhuo Xu

1 Introduction

The Human Immunodeficiency Virus (HIV) has been a major global health problem for decades. Researchers working in HIV related fields has successfully developed some treatment methods. A common used therapy combines two nucleoside reverse transcriptase inhibitors (NRTI) with a protease inhibitor (PI) [4]. When these inhibitors are absorbed into a cell infected with the HIV virus, they can inhibit viral replication and stop the virus from infecting other cells. These proposed therapy methods can be simulated by dividing treatment into discrete time steps and modeling on how states of those cells change. Simulations rely on rule sets according to cellular automata (CA) to produce reliable cell states in later stages. CA model is a simulation model commonly used to model a large number of cells and their interaction. Many researchers apply CA to simulate the dynamics of HIV infection [9]. However, simulations for these therapy methods cost a huge amount of computing power and take much time to generate results.

In this paper, we bring machine learning techniques to predict cells' state at the end of 300 weeks treatment. Based on the data from a large amount of implemented simulations, we developed several machine learning models that can predict the cell states at week 600 on the basis of the features from the first 300 weeks' cell states. The prediction can be used to evaluate the effectiveness of therapies without spending time on running simulations for each case. This kind of prediction is very useful on investigating the effectiveness of therapy methods and predicting the percentage of cells to remain healthy in the future.

The organization of this paper is as follows. In section 2, we describe how the HIV models work with cellular automata concept. In section 3, we present the results of our machine learning models and the description of our regressors. Section 4 is the discussion of results. Section 5 is the conclusions we reached from our findings.

2 Methods

For this research, *cellular automata* (CA) reproduced the effects of Human Immunodeficiency Virus (HIV) within the human body. There are five implementations of HIV modeling used: the dos Santos, González, Moonchai, Precharattana, and Rana models, which will be further discussed in section 2.1. From several thousand simulations of these models at various sizes (side lengths of 800×800, 1000×1000 and 1200×1200), image features were collected every five timesteps until $t = 200$. These image features are described in detail in section 2.2. Once this step of data collection was complete, several regression models were used to predict the prevalence of each cell type at the end of the simulation for each of the HIV models; these regression models included the *Decision Tree Regressor* (Tree), *Gaussian Processes Regressor* (GPR), *Lasso Regressor* (Lasso), and *Support Vector Regressor* (SVR). The regression models and *hyperparameter tuning* (HT) are described in section 2.3.

2.1 HIV Modeling using CA

Researchers have worked on developing virtual laboratories using CA for about 30 years, dating back to the model of Kougias and Schulte in 1990 [6] and a new spur of research activity following the work of dos Santos and Coutinho in 2001 [12]. Many such models are discussed in the 2016 review of CA for HIV by Precharattana [9], and some are briefly presented in a newer survey addressing the broader context of applied CA research [1]. Noteworthy examples include a 2010 model by Precharattana [10] which introduced the notion of "reservoirs" (i.e. cells in which HIV can hide for years), a 2013 model by González, et. al, which accounted for two classes of drugs [4], a model by Rana, et al., presented at the ACM SIGSIM PADS'15 on the effect of lapse in treatment [11], a 2016 model by Moonchai and Lenbury experimenting on the effects of giving drugs and replacing a patient's blood plasma [8], and an ACM SIGSIM PADS'19 piece seeking to model two modes of HIV transmission between cells [2].

Each of these HIV models use a form of CA. A cellular automaton is a simulation model composed of large numbers of simple components (called cells) which only interact with a small set of other components. These local interactions are defined as a neighborhood. In most HIV models apart from [7], cells are packed within a regular 2D lattice, that is, a grid. Each cell of the cellular automata simulation model represents a CD4+ T

immune system cell in the human body. Within this grid, the neighborhood of a cell consists of the four cells that touch via a side (Von Neumann neighborhood) or also includes the other four cells that touch via a corner (Moore neighborhood). HIV CA models generally use a Moore neighborhood, although recent investigations suggested that neighborhoods of more than eight cells may be needed to capture different ways in which the HIV virion spreads [2]. Although cells are packed in a grid, the boundaries (e.g., first row, last column) are a special case because cells at such locations have fewer neighbors, which can introduce unwanted irregularities into a simulation. All HIV CA apart from [11] dealt with this special case by wrapping boundaries (figure 1) from one side onto the other (e.g., the top-left cell now

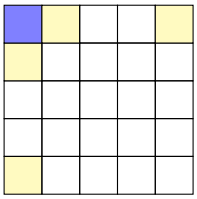

Figure 1: Example of "wrap around". Von Neumann neighbors of the top-left cell are highlighted in light yellow.

also connects with the top-right cell and the bottom-left cell).[1]

Each cell within the grid is given a state (healthy, infected, dead, or some other intermediate or special states, such as acute infected, healthy with treatment present, etc.) based on a particular ruleset of the simulation model. This process of kickstarting a CA simulation is called seeding, as the following steps of the model require a starting point to begin simulations. The model then uses the aforementioned ruleset to update each cell synchronously over discrete time steps based on its state and the states of cells within its neighborhood. "Discrete" in this context means that the entire grid is updated at one time for each time step. The time step itself represents the passage of a period of real-life time, which is weeks in the case of these HIV simulation models. Each model ends at 600 timesteps, which is 600 weeks of HIV simulation. An example of discrete time stepping is shown in figure 2, where 2-D cellular automaton evolved 3 times. The rule set of this CA is given below the time-evolution.

Finally, it must be noted that there were modifications to the CA HIV models used by this team for the purposes of increasing performance. Based on previous findings [3], each model was modified to make use of Just In Time (JIT) compilation, parallel processing, Xoroshiro128+ Pseudo-Random Number Generation (PRNG), and neighborhood addition, which treats each state as a numerical value rather than a discrete class. These modifications allow for far faster processing of each full simulation run, allowing for more data to be collected in a shorter period of time.

2.2 Image Feature Collection

After running each HIV simulation model 300 times at the three desired grid lengths, all of the timestep data was collected for image feature analysis. Each grid for each timestep can be viewed itself as a greyscale image. Using the SciKit-Image and OpenCV libraries, image features can be recognized in these greyscale images of each timestep. Once these features were recognized and labeled by the image libraries, a numerical value must be pulled from them to regress on. This poses a challenge: *how does one turn the location of a feature into a numerical value?* For the purposes of this paper, this team focused on easily obtainable numerical values such as the number of occurrences of a feature or the average distance between said occurrences. In particular, given the nature of the HIV CA models, three features in particular stand out: blobs, corners, and contours. The image features described below were identified at timesteps $t \in \{0, 5, \ldots, 195, 200\}$.

Blobs can be described as large masses of similarly colored cells. figure 3 demonstrates what a blob looks like within one of the HIV CA models. Once blobs have been located in the image using the various libraries, a few values were extracted. From the OpenCV library, blobs features including the number of blobs with an area of at least 10, the average distances between any two blobs with an area of at least 10, the average size of blobs with an area of at least 10. From the Sci-Kit Image library, blob features included

- number of blobs located by `LogBlob`,
- average size of blobs,
- variance in the x coordinates of located blobs,
- variance in the y coordinate of the located blobs.

Contours can be described as the inner or outer edge of a shape formed by similarly colored cells. This shape, unlike a blob, is not filled in with uniformly colored cells, but contains completely different colors within its interior. An example demonstrating a contour can be found in figure 4. After locating the contours using the OpenCV library, the following values were obtained:

- number of contours with an area of at least 10,
- avg. area of contours with an area of at least 10,

[1] Let it be noted CA grids can be of different shapes such as a triangle or hexagon, and the cells can also be of different sizes. CAs are usually taught as a 2-D shape but they can be reformatted to the nth dimension such as 1-D or 3-D.

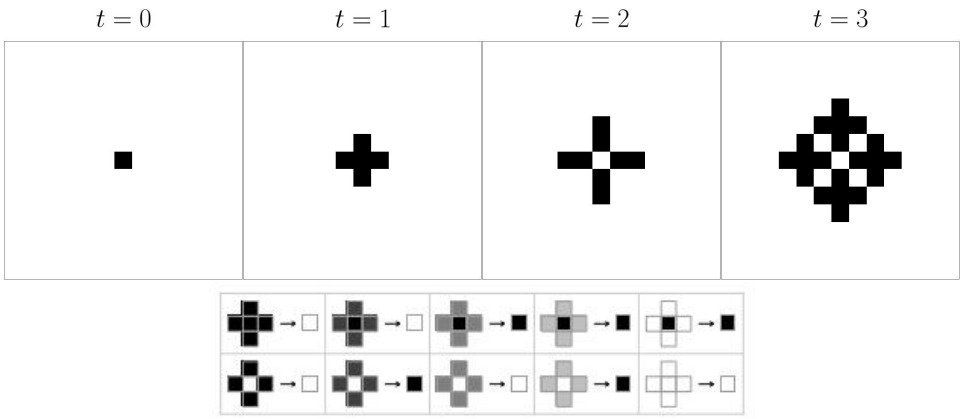

Figure 2: Example cellular automaton evolving by 3 time steps. The rule set of this CA is given below the evolution. In this rule set, each cell can either be black or white, but the shades of gray of the 4 surrounding cells indicate the number of black neighbors around the center cell. When its neighbors are completely white, there are no black neighbors around the center cell. Subsequent darker shades of gray indicate increasing number of black neighbors $(1, 2, 3, 4)$ around the center cell.

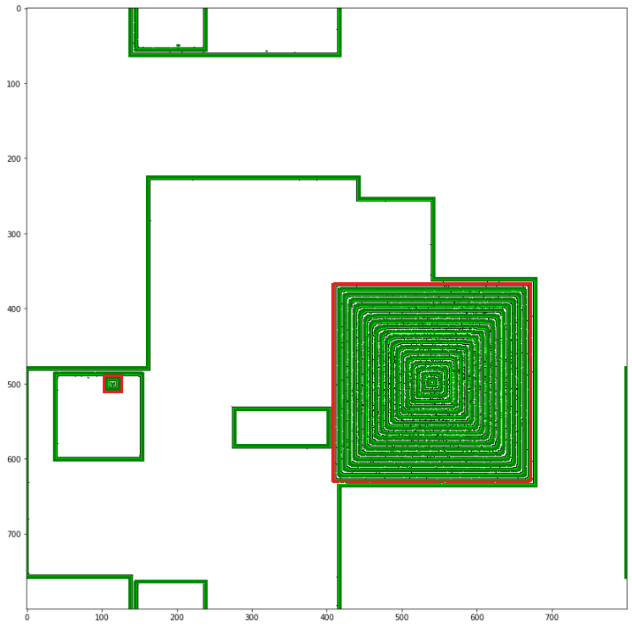

Figure 3: Blobs (highlighted by red borders) that can be detected by image processing libraries.

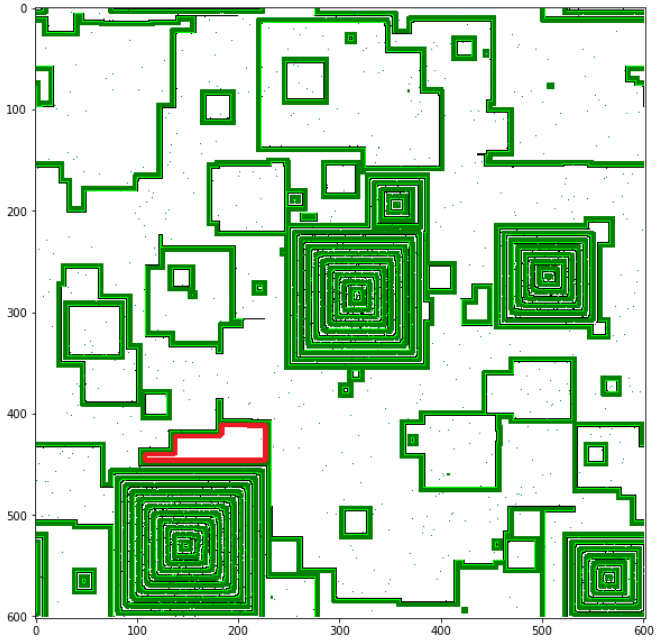

Figure 4: Contours (highlighted by red borders) that can be detected by image processing libraries.

- number of contours with an area of at least 100,
- avg. area of contours with an area of at least 100,
- number of contours with perimeter of at least 10,
- avg. area of contours with perimeter of at least 10,
- number of contours with perimeter of at least 20,
- avg. area of contours with perimeter of at least 20,
- maximum area of all contours,
- maximum perimeter of all contours.

Corners can be described as a sharp turn taken by a strip of similarly colored cells. figure 5 shows the occurrence of several corners recognized by the Sci-Kit Learn library. Using this library, following values were obtained:

- number of corners found by the Harris function,
- distance between found corners.

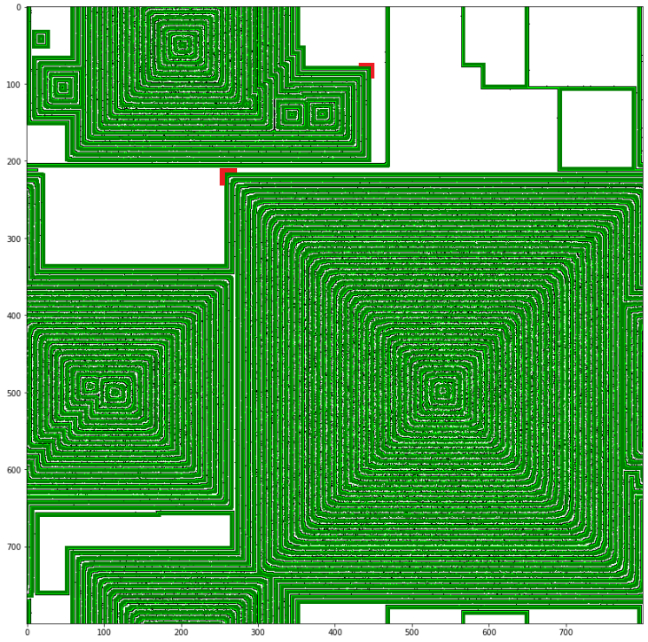

Figure 5: Corners (highlighted by red borders) that can be detected by image processing libraries.

2.3 Regression

With the image features collected, the data is nearly ready for regression. *Regression* is the statistical practice of predicting some numerical measure of a new data point in a dataset, given the characteristics of said data point. The target measures, or *labels*, for the HIV CA models are the percentages of each target class (healthy, dead, latently infected, and acute infected). From each of the simulations runs of all five HIV models, all 4 class percentages were calculated at the final timestep, $t = 600$, or 600 weeks after the onset of HIV in the human body. Once the composite data set of image features and target labels is created, regression can begin. The regressors choose a target label to predict, and are fed increasingly more information; at first, they have access to the data from $t = 0$, then from both $t = 0$ and $t = 5$, and so on until the regressor has access to the data from all timesteps during which image information was collected. At the end of each regression task, an error rate is calculated using *root mean squared error* (RMSE), which, for a sequence of data $\{y_i\}_{i=1}^{N}$, is given by

$$\text{RMSE} = \sqrt{\sum_{i=1}^{N} \frac{(y_i - \bar{y})}{N}},$$

where $\bar{y}$ is the sample average. The RMSE is a positive value that dictates how far a regressors prediction is away from the target label. For instance, if the target label indicated that 7.3% of the cells at the end of a simulation run were dead, and a regressor predicted that 6.1% of the cells would be dead given information from timesteps 0 through 100, an RMSE value of 1.2. In context, this would mean that this particular regressor will predict, on average, a percentage of dead cells that is off from the true percentage by 1.2%.

The goal of regression is to lower the RMSE error without approaching an error of 0. If this were to happen, then the model would not adapt well to new data points, and would be *overfit* to the current dataset. Thus, a rigorous approach must be taken to lower error without overfitting. This process is a combination of *nested cross-fold validation* (CFV) and *hyperparameter* tuning (HT). When training a regression model on a dataset, there must be a subdivision within the dataset to *test* the model on. It cannot be tested on data it has already seen; rather, the test set of data is not seen by the model until it is being tested for accuracy. This, to properly validate the accuracy of a model, CFV is utilized. This process breaks down the dataset into k folds or "chunks". The model is first trained on the first $k-1$ folds, and tested on the kth fold. This process is repeated until all k folds have been utilized as the test set of data. The *nested* portion of this process comes from further subdividing the dataset one more time into a *validation set*. This validation set is a subset of the training set already previously identified. Why validate the model before testing it? This comes back to HT. Hyperparameter tuning is a process by which the various parameters of a model are tuned to have the lowest error. In order to find which given set of parameters obtains the lowest error, each regressor must be validated. Listed below are the various models used and the hyperparameters that were tuned:

- **Support Vector Machine** This regressor used two different core functions, called kernels, to calculate its regressions. These kernels are the Radial Basis Function (RBF) and Polynomial kernels of degree d

  $$\begin{aligned} &\text{RBF} && K(\mathbf{a},\mathbf{b}) = \exp\left(-\gamma\|\mathbf{a}-\mathbf{b}\|^2\right), \\ &\text{Polynomial} && K(\mathbf{a},\mathbf{b}) = (\gamma\mathbf{a}^T\mathbf{b}+r)^d. \end{aligned}$$

  Sci-Kit Learn allows for many parameters in its SVR function, but our team found after testing that the `gamma`, `C`, `epsilon`, and `degree` values showed the most promise for getting low errors [5].

- **Gaussian Process** Much like the SVR, the GP regressor also has its own kernels. These are the RBF and Rational Quadratics (RQ) kernels. After preliminary testing, the `alpha` and `length_scale` parameters also showed more promise than others.[2]

[2]The computational cost of the RQ kernel is significantly higher

- **Lasso** Unlike previous regressors, this regressor does not have different kernels. Instead, the only viable hyperparameter is `alpha`.

- **Decision Tree** Parameters `max_depth` and `min_samples_split` were tested.

Once all parameter combinations have been validated within the *inner fold* (the fold the CFV with the validation sets), then the regressor with the best parameter combination moves to the *outer fold* to calculate an average error and standard deviation from this average error using the k folds of test datasets. The results from this rigorous process are explored in Sections 3 and 4.

3 Results

The objective of this section is to show how the different regressors were at predicting the final state of each model based on the features extracted. Python was used during the entire process of simulating each model, extracting features at each time step, and running the four regression algorithms. Before this, each model was simulated to understand how the different algorithms affected the final result. In the following sections, we first examine the results from running each model. Second, we examine the feature extraction algorithms used and lastly examine the plots generated after running the four regressors.

3.1 HIV Modeling

Every model started the same way with mostly healthy cells and 5% infected cells. After $t = 0$, the number of infected cells increases by infecting the healthy cells. All the models went through some latent infected stage where the number of infected cells increased rapidly before entering a second phase where the number of infected cells increased slowly. Each model defines different types of infected cells which affected each model differently. The different algorithms had different numbers of infected cells at the last time step with the Rana model performing the best with only about 275000 infected cells. The Gonzalez model results had zero infected cells but there was an error in the logic of the program which caused it to have inaccurate results. The other two models, Precharattana and Dos Santos had about 450000 and 455000 infected cells respectively. These results are based on a grid size of 1000×1000.

than the other regressors examined, and did not finish running in time. Only results from the RBF kernel is obtained and studied at the time of writing, but we plan on merging the two kernel's data as soon as they become available.

3.2 Feature Extraction

After each model was simulated and a grid was saved for every time step, features were extracted based on what was observed. As the number of infected cells spread, this affected the number of features detected during the feature extraction process increased. There were 20 features extracted such as the number of blobs with certain areas, corners and contours. A more specific detail about the features extracted were discussed in the methods above. When the $t = 0$, there were no features detected but as more time passed, the number of features detected increased. When observing a grid after 100-time steps, a ripple effect was observed. For every time step afterwards, the ripple effect increased in frequency until the grid looked like random noise. The number of blobs also increased as the time step increased.

3.3 Regression Plots

Once the features were extracted from the simulations, they were run through different regressors to see if the last time step could be predicted. One thing to note is that the preferred parameters used for each regressor was not consistent from the beginning of the simulation to the end. When looking at the decision tree regressor for the Gonzalez model, the best fit parameter values for max depth and min samples vary from $t = 0$ to $t = 200$. This is consistent throughout all the models where the best parameters do not stay the same from the beginning to the end of the simulation. After running the regressors, the data that was generated was examined and then plots of the RMSE were examined to see how each regression algorithm performed. Each cell count was converted into a percentage before being fed into each regression algorithm.

3.3.1 Support Vector Machine

The first regressor looked at is the support vector machine shown in figure 6. For the Dossantos graph, the RMSE for healthy cell plot started around 0.02 and stayed the same for every timestep. This is the same for the dead and infected cells. From examining the plot further, it can also be seen that the RMSE for the A1 infected cell plots has the highest variance while dead and A2 infected cell plots have the smallest variance. For the Gonzalez model, the RMSE plots also stayed about the same throughout the simulation. The RMSE variance for the A1 infected cells and the healthy cells has a lot of overlap and the A2 infected cell plots has a lot of overlap with the dead cell plots. The graph for the Rana graph follows a similar trend to the last two and the Precharattana graph mostly follows a similar trend to the other graphs except for the start and middle of the

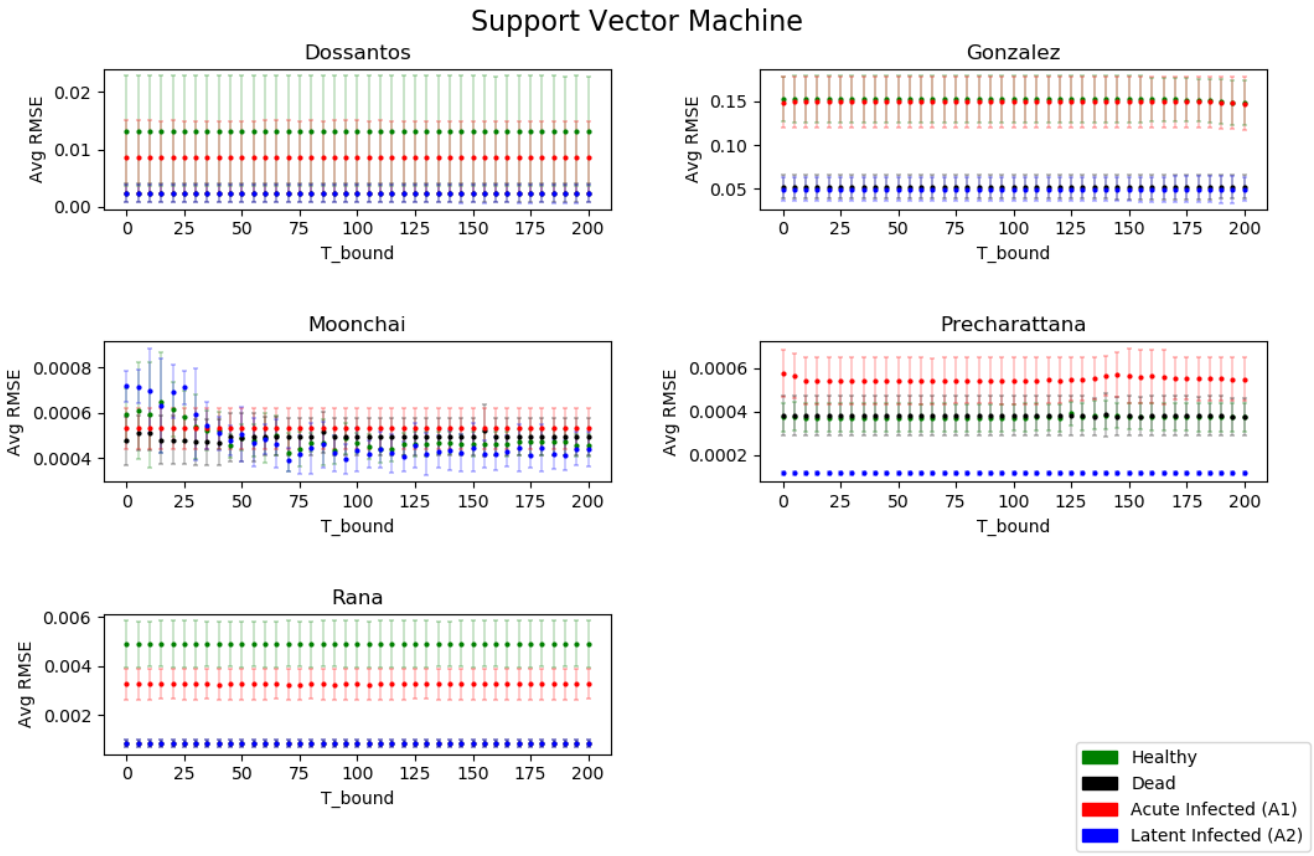

Figure 6: RMSE of the support vector machine regressor for data from each CA model after a 10-fold cross validation. The y axis plots the percentages in decimal, i.e. 0.01 means 1% (of total cells count). The x axis plots the maximum t up to which image features were considered by the regressor. A_1 denotes acutely infected cells, and A_2 denotes latently infected cells.

simulation. During the first couple time steps, the variance decreases, increases again in the middle of the simulation and then decreases again. The Moonchai graph showed the most change than any other graph, mainly for the A2 infected and healthy cell plots. The two plots follow a decaying sinusoidal pattern as the time step increased. Another thing to note is that the Moonchai and Precharattana graphs had the smallest RMSE with under 0.0008 and 0.0006 respectively while Gonzalez had the largest RMSE at 0.15.

3.3.2 Gaussian Process

The second regressor examined is the gaussian process regressor shown in figure 7 which has the same range for the RMSE as the SVM regressor. One notable exception is in the first couple time steps of the Moonchai graph. At the start of the simulation, the calculated RMSE for the A2 cells is around 0.0005 and decreases for the next couple timesteps. As the simulation continues and the number of A2 cells increases, the calculated RMSE at time step 25 jumps up to 0.0010 and has a larger variance. The RMSE for the A2 infected cells and the A1 infected cells is different than the SVM regressor in that the A2 infected cell RMSE plot is higher than the RMSE plot for the A1 infected cells. This is only true for the time steps after 30 because before this the RMSE for Moonchai's A2 cells was higher than 0.0006. The RMSE for the Precharattana and Gonzalez graphs is similar to the SVM regressor. The only difference in the Gonzalez graph of this regressor is that there is a slight drop in the start of the graph until time step 30 when it goes back to the starting value. The Rana graph has a maximum RMSE of about 0.006 and stays at around the same RMSE through the entire graph. There is one drop in the RMSE at time step 15 but jumps back up to the previous RMSE at time step 25. Except for the Moonchai plot, the RMSE is greater for the healthy and A1 infected cell plots and lower for the A2 infected and dead cell plots.

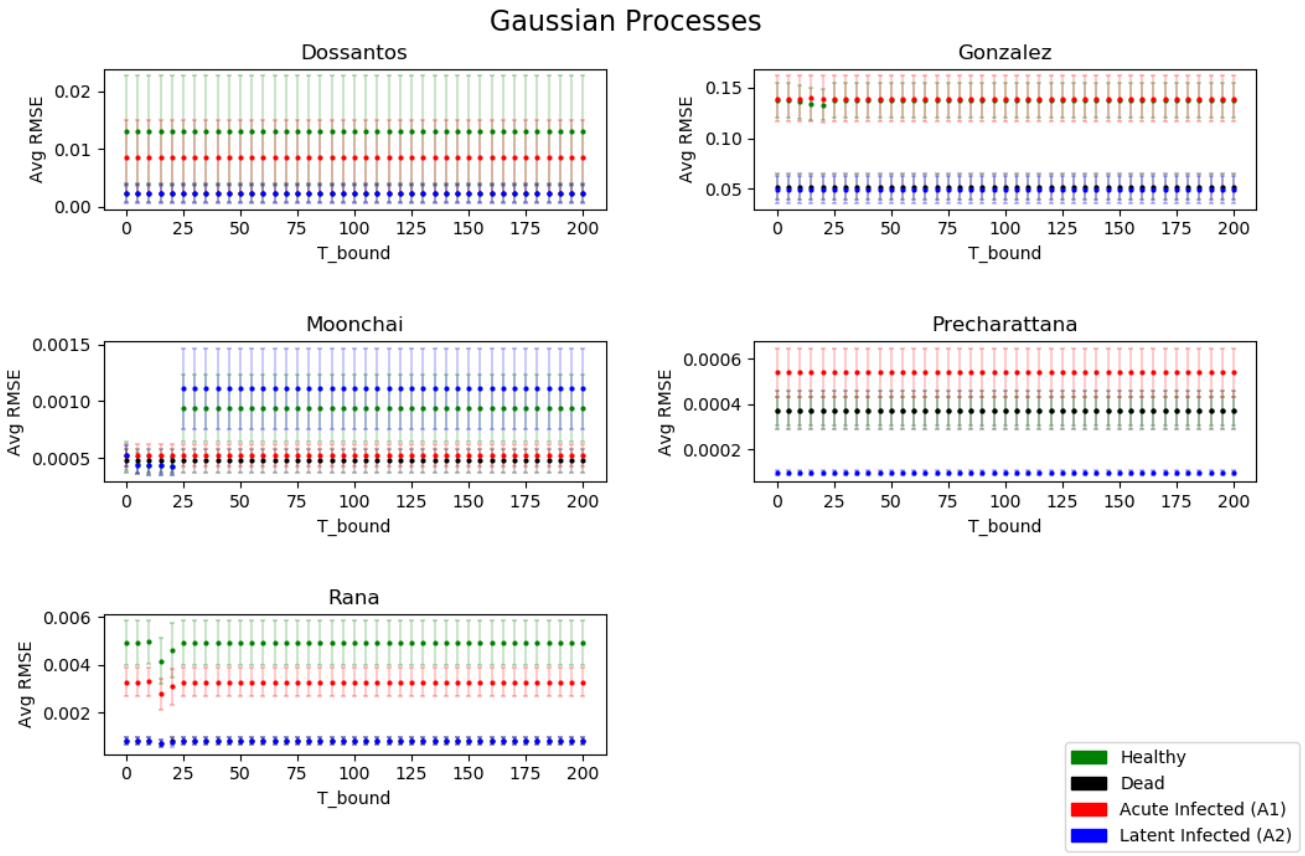

Figure 7: RMSE of the gaussian process regressor for data from each CA model after a 10-fold cross validation. The y axis plots the percentages in decimal, i.e. 0.01 means 1% (of total cells count). The x axis plots the maximum t up to which image features were considered by the regressor. A_1 denotes acutely infected cells, and A_2 denotes latently infected cells.

3.3.3 Lasso

In the lasso regressor shown in figure 8, the ranges for the graphs are roughly equivalent to the last two regressors except for the Moonchai graph for the Gaussian process. The Gonzalez graph has the highest RMSE and the Moonchai and Precharattana graphs have the smallest RMSE. In the Precharattana model, the RMSE for the A2 cells is 0 throughout the whole simulation. The Rana graph with this regressor has the most fluctuations even though it ends and begins with the about the same RMSE and variance. The graph shows that the regressor got the least accurate at predicting the outcome around the 25th timestep but got gradually better afterwards. The A2 and the dead cell plots were two cell types that had the smallest RMSE in all the graphs while the A1 and the healthy cells had the largest RMSE. In the Dossantos and the Rana graphs, the healthy cells had the largest RMSE while in the Precharratana, Moonchai and Gonzalez graphs, the A1 cell plots had the highest RMSE.

3.3.4 Decision Tree

The last regressor examined is the decision tree regressor shown in figure 9 with the RMSE being plotted for each time step in each model. The biggest thing that stands out is that Gonzalez has the largest RMSE just like the other graphs. The plots with the smallest RMSE plots are Mooncai and Precharattana. When looking at the five graphs, it can be seen that the A2 infected cells plots have the smallest RMSE and variance than other three cell types. The RMSE for each graph stays about the with some fluctuation throughout the simulation while the variance either increased or decreased for each graph. The Rana graph has a largest fluctuation in the beginning but decreases and levels off at about the 20th time step. The Dossantos graph has the opposite effect where it starts with a smaller RMSE and increases for the next couple time steps. It has another spike in its RMSE at around time step 105 and 110.

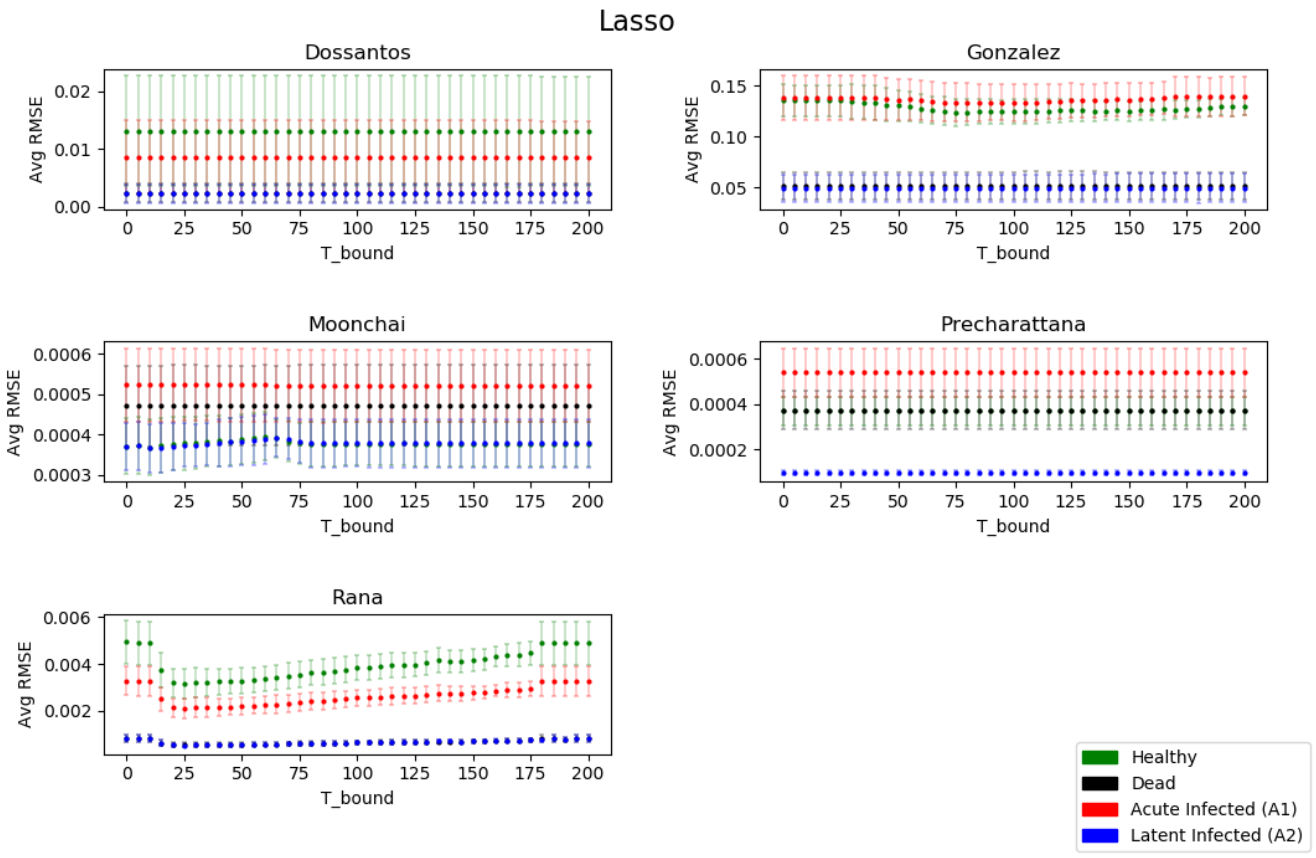

Figure 8: RMSE of the lasso regressor for data from each CA model after a 10-fold cross validation. The y axis plots the percentages in decimal, i.e. 0.01 means 1% (of total cells count). The x axis plots the maximum t up to which image features were considered by the regressor. A_1 denotes acutely infected cells, and A_2 denotes latently infected cells.

4 Discussion

HIV virtual laboratories have existed for a quite a long time, as has using cellular automata to simulate a small group of cells. The problem with simulating disease dynamics in an individualized manner for a patient is how computationally expensive it may become. Many patients do not have the access to the resources necessary to run these simulations. With a large grid size, many time steps, and variability requiring up to thousands of replications to guarantee 95% confidence, even a minor increase in cell count or grid size can have a more pronounced effect on the computational requirements of the simulation. There is a balance to be struck between lowering these requirements and providing results that are accurate enough to be confidently used in the decision-making process for treatment. The process detailed within this paper describes an approach that attempts to find this balance using feature extraction and regression.

4.1 Present Work

The first step in this procedure was to identify five different models, each with their own ruleset to determine cell state-changes, and seed each model. The models were each optimized if they did not contain previously researched optimization methods such as JIT, parallel processing, and PRNG. Each model was run 300 times for each grid size, with data being collected and viewed as greyscale images with the SciKit-Image and OpenCV libraries. Feature extraction was performed on desired timesteps up to 200. Then Support Vector Regressor, Gaussian Processes, Lasso, and Decision Tree regressors were each used to get the error rate with RMSE. To avoid overfitting the new model to the data, nested cross-fold validation and hyperparameter tuning were employed.

4.2 Implications of Results

Based on the results there is clear evidence that the predictions are viable to a degree, but there are some

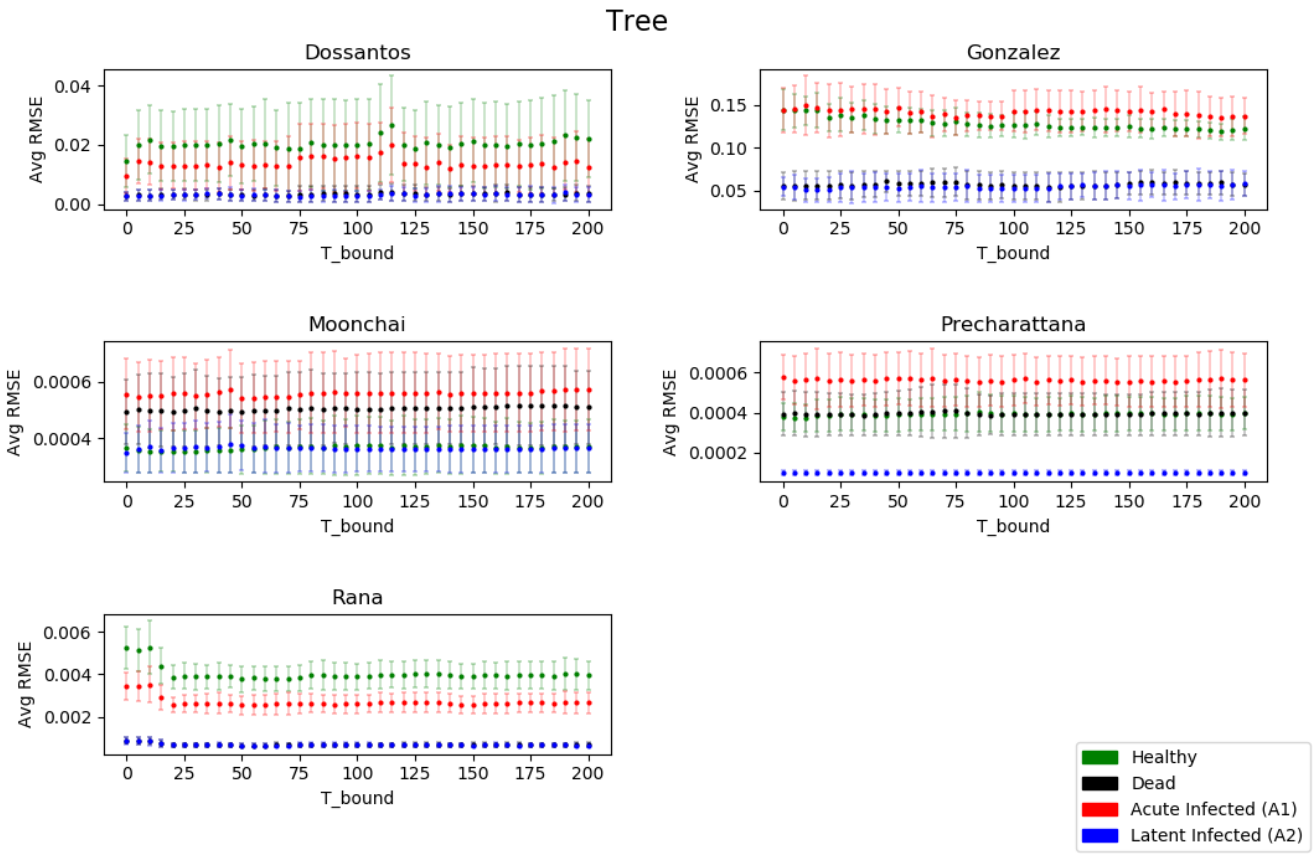

Figure 9: RMSE of the decision tree regressor for data from each CA model after a 10-fold cross validation. The y axis plots the percentages in decimal, i.e. 0.01 means 1% (of total cells count). The x axis plots the maximum t up to which image features were considered by the regressor. A_1 denotes acutely infected cells, and A_2 denotes latently infected cells.

concerns. The RMSE values for each plot vary only slightly throughout the timesteps and for each regressor, but a few of the results are very promising. RMSE for Moonchai and Precharattana were 0.0008 and 0.0006 for the state vector machine. Gonzalez RMSE struggled in both the SVR and Tree regressors, sitting at around 15%. Moonchai and Precharattana routinely have the lowest RMSE's regardless of regressor used, while Gonzalez tends to have the highest. Despite generally low RMSE's, there is little dependence on the RMSE when compared with t-Bound. There were some spikes, but the trend for each RMSE generally stays the same. This was unexpected and suggests that the features used were not the most informative. This implies that a better set of features would benefit the model. Some class labels were consistently low and their validity is questionable at best. The only regressor/model combination that achieved expected results was the Rana model with the Tree regressor.

4.3 Limitations

Despite measures being taken to improve upon the original simulation process, there were a few limitations with this proposed process. First, there are additional features that may not have been explored. In this paper most features were focused on finding blobs and their characteristics. Some features relied on corners or contours, but there are many more possibilities when it comes to feature extraction. It is possible that there are feature combinations that have not been tested that yield statistically better results. Of the features that were collected, some of the complex information about these features was not investigated, such as the orientation of corners, or taking clustering into account. Additionally, the images could have been treated as boolean masks. In the case of boolean masking, each classification would be treated as a true/false value, allowing us to easier quantify the cell counts.

The four regressors took days to run despite effective use of the high-performance computing cluster. Time

will likely always be a limitation regardless of the ability of the machine running these simulations. Our preliminary testing did not allow us to include many of the hyperparameters because they showed a lack of promise. The coef0 parameter for the SVR regressor and some of the Gaussian Processes kernels fell into this category. Additionally, these parameters were each only tested with a small number of values, between two and four. As a result of the testing conducted, some hyperparameters displayed such an adverse effect on run-time that they had to be excluded.

Specifically, for the Gonzalez model the treatment process did not start until timestep 300. Features were only extracted for timesteps up to 200. This model was expected to have high variability (citation/specific evidence here), but due to the timestep restraints error was as large as 15% after regression, markedly higher than the other models. Since the method predicts the end-state of the model using the image features, predicting the end state using only time states up to and including $t = 200$ does not return an accurate result. At this stage of the simulation, the Gonzalez model did not have a chance to deploy its treatment and garner independent and accurate results. Predicting the end state based off of states where treatment had not taken effect simply is not accurate.

This method does suggest an improvement over the initial simulations but based on the limitations there is room for further improvement. As previously mentioned, there are combinations of features that have not been explored in this procedure. Pooling together a more robust set of diverse features including those outlined above along with edge and corner-focused features could improve results. Exploring the use of boolean masks may have made the computational requirements of these models slightly less intensive, saving time and allowing the inclusion of a few more parameters. Using a larger span of values to test the hyperparameters may indicate that some of these parameters are viable to include in the regression. During model validation, implementing bins could allow for use of stratified k-means to improve our model validation scores.

5 Conclusion

We investigated the possibility of predicting the prevalence of HIV 10 years after initial infections, based on image features extracted from simulated snapshots of the host's lymphatic tissue at various stages of the infection using different CA models. Although dependent on the specific CA model, the generally low error rates — even when considering features up to relatively small t — demonstrate the practicality of such predictions using machine learning algorithms. Due to the low computational cost of future predictions after training, this technique can help physicians provide effective and personal HIV treatment plans, thereby increasing patients' quality of life. The present results suggest there might be more informative features to extract as the dependence of accuracy on t-bound is weak. Therefore, future work should comprehensively examine image features that are more closely tied to the underlying biological processes, as well as tuning simulation parameters to more accurately reflect real-life viral dynamics to further improve the accuracy of predictions.

References

[1] Kamalika Bhattacharjee, Nazma Naskar, Souvik Roy, and Sukanta Das. A survey of cellular automata: types, dynamics, non-uniformity and applications. *Natural Computing*, Jul 2018.

[2] Philippe J. Giabbanelli, Cole Freeman, Joshua A. Devita, Nicholas Rosso, and Zabrina L. Brumme. Mechanisms for cell-to-cell and cell-free spread of hiv-1 in cellular automata models. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '19, page 103–114, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Philippe J. Giabbanelli, Jared A. Kohrt, and Joshua A. Devita. Optimizating Discrete Simulations of the Spread of HIV-1 to Handle Billions of Cells on a Workstation. Unpublished.

[4] Ramon Gonzalez, Sergio Coutinho, Rita Zorzenon dos Santos, and Pedro Figueirêdo. Dynamics of the hiv infection under antiretroviral therapy: A cellular automata approach. *Physica A Statistical Mechanics and its Applications*, 392:4701–4716, 10 2013.

[5] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification, 2003.

[6] Ch. F. Kougias and J. Schulte. Simulating the immune response to the HIV-1 virus with cellular automata. *Journal of Statistical Physics*, 60(1):263–273, Jul 1990.

[7] Youbin Mo, Bin Ren, Wencao Yang, and Jianwei Shuai. The 3-dimensional cellular automata for hiv infection. *Physica A: Statistical Mechanics and its Applications*, 399:31 – 39, 2014.

[8] Sompop Moonchai and Yongwimon Lenbury. Investigating combined drug and plasma apheresis therapy of hiv infection by double compartment cellular automata simulation. *International Journal of Computer Theory and Engineering*, 8:190–197, 06 2016.

[9] Monamorn Precharattana. Stochastic modeling for dynamics of HIV-1 infection using cellular automata: A review. *Journal of Bioinformatics and Computational Biology*, 14(01):1630001, 2016. PMID: 26620040.

[10] Monamorn Precharattana, Wannapong Triampo, C. Modchang, and Yongwimon Lenbury. Investigation of spatial pattern formation involving CD4+ T cells in HIV/AIDS dynamics by a stochastic cellular automata model. *International Journal of Mathematics and Computers in Simulation*, 4:135–143, 01 2010.

[11] Ela Rana, Philippe J. Giabbanelli, Naga H. Balabhadrapathruni, Xiaoyu Li, and Vijay K. Mago. Exploring the relationship between adherence to treatment and viral load through a new discrete simulation model of hiv infectivity. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '15, page 145–156, New York, NY, USA, 2015. Association for Computing Machinery.

[12] Rita Maria Zorzenon dos Santos and Sérgio Coutinho. Dynamics of hiv infection: A cellular automata approach. *Phys. Rev. Lett.*, 87:168102, Sep 2001.