Predicting Mutational Pathways of Influenza A H1N1 Virus using Q-learning

Aarathi Raghuraman

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Lenwood S. Heath, Chair
T. M. Murali
Boris A. Vinatzer

July 22, 2021
Blacksburg, Virginia

Keywords: Fitness, Mutational Pathways (or Paths), Reinforcement Learning, Q-Learning

Predicting Mutational Pathways of Influenza A H1N1 Virus using Q-learning

Aarathi Raghuraman

(ABSTRACT)

Influenza is a seasonal viral disease affecting over 1 billion people annually around the globe, as reported by the World Health Organization (WHO). The influenza virus has been around for decades causing multiple pandemics and encouraging researchers to perform extensive analysis of its evolutionary patterns. Current research uses phylogenetic trees as the basis to guide population genetics and other phenotypic characteristics when describing the evolution of the influenza genome. Phylogenetic trees are one form of representing the evolutionary trends of sequenced genomes, but that do not capture the multidimensional complexity of mutational pathways. We suggest representing antigenic drifts within influenza A/H1N1 hemagglutinin (HA) protein as a graph, $G = (V, E)$, where V is the set of vertices representing each possible sequence and E is the set of edges representing single amino acid substitutions. Each transition is characterized by a Malthusian fitness model incorporating the genetic adaptation, vaccine similarity, and historical epidemiological response using mortality as the metric where available. Applying reinforcement learning with the vertices as states, edges as actions, and fitness as the reward, we learn the high likelihood mutational pathways and optimal policy, without exploring the entire space of the graph, G. Our average predicted versus actual sequence distance of 3.6 ± 1.2 amino acids indicates that our novel approach of using naive Q-learning can assist with influenza strain predictions, thus improving vaccine selection for future disease seasons.

Predicting Mutational Pathways of Influenza A H1N1 Virus using Q-learning

Aarathi Raghuraman

(GENERAL AUDIENCE ABSTRACT)

Influenza is a seasonal virus affecting over 1 billion people annually around the globe, as reported by the World Health Organization (WHO). The effectiveness of influenza vaccines varies tremendously by the type (A, B, C or D) and season. Of note is the pandemic of 2009, where the influenza A H1N1 virus mutants were significantly different from the chosen vaccine composition. It is pertinent to understand and predict the underlying genetic and environmental behavior of influenza virus mutants to be able to determine the vaccine composition for future seasons, preventing another pandemic. Given the recent 2020 COVID-19 pandemic, which is also a virus that affects the upper respiratory system, novel approaches to predict viruses need to be investigated now more than ever. Thus, in this thesis, I develop a novel approach to predicting a portion of the influenza A H1N1 viruses using machine learning.

Dedication

Dedicated to Amma and Appa.

Acknowledgments

First, and foremost, I want to thank my advisor, Dr. Lenwood Heath for being flexible, compassionate, patient, and an unwavering support throughout the Master's Degree. He worked with me patiently to find a topic at the intersection of our interests, guided me through the literature review and provided me constant encouragement during the ups and downs of COVID-19, the research process and the job search. I have learned a lot of professional and personal skills from Dr. Heath that will last me a lifetime. I also, greatly appreciate the knowledge and insight that Dr. T.M. Murali and Dr. Boris Vinatzer provided as a part of my committee. Their critical feedback and questions have been vital to the thesis. The Computer Science Department, especially Dr. Cliff Shaffer, have been especially supportive by listening and addressing graduate student concerns throughout the pandemic. Dr. Cliff Shaffer took the time to ask students how they were doing at every meeting he had with students to ensure their well-being at all times. His open office hours and quick responses were essential to my continued progress through graduate school.

I am extremely grateful to the Center for the Enhancement of Engineering Diversity (CEED) for providing me a Graduate Teaching Assistantship during the two years of my Master's degree. I have thoroughly enjoyed empowering freshmen female engineers to pursue their dreams without hesitation. Thank you Dr. Bevlee Watford, Susan Arnold-Christian, Dr. DeAnna Katey, Dr. Kim Lester, Becky Shelor and Taylor Cupp for welcoming me to CEED each year. Thank you to Kayla Chauffin, Trey Waller and Renee Cloyd for always being resourceful. Thank you to the GTA crew: Conor, Callie, Becca, Awad, Dominic, Mekonen, Jasmine, Carol, Sophia, Alex, Perry, Daniela, Taylor, Hannah, Bemnet and Oreoluwa for an incredible time during and after work.

To my friends, thank you for the encouragement and support throughout my thesis. Nure and Blessy thank you for being the best library accountability buddies and for the fun study breaks. Thank you to the writing group: Rachel, Dominic, Wenchuo, Mekonen and Alex for pushing me to finish the thesis. Another shout out to Rachel for the rapid and valuable edits on the thesis and the defense presentation. Without her help, the thesis would have been a grammatical and logical mess. Thank you, Renee and Nathan for your continued friendship since undergraduate and for sharing your kitties Sirius, Tobby, Mister, and Raleigh.

Thank you to the most emotionally supportive network that I am blessed to call family. Andrea and Dan, thank you for the food runs, laundry help, hugs and love. You have provided me a home away from home in Blacksburg. Thank you Gerriann, Owen, Stevie, Avinash, Swetha and Akshara for the many video calls and visits. Thank you Amma and Appa for the late night zoom calls, constant encouragement, many presentation runs, venting sessions, hugs and love, without which I would have quit after the first semester. Amma and Appa you taught me the value of smart work and pushed me to pursue a Master's and I cannot express how grateful I am for that. Thank you Ammumma for your blessings and confidence in me when I wavered. Your kind words always lifted me up. Thank you to the rest of my family for the fun diversions through texts and calls.

Lastly, thank you to the love of my life, my cheerleader, the light during darkness, my beloved partner, Conor. Thank you for always being there no matter how hangry, cranky or grumpy I was. Thank you for the much needed walks that I always refused, the long thought-out surprises, the many philosophical conversations, the unasked food runs for delicious freshly squeezed orange juice, the news updates to keep me informed, the long days and nights on campus, the multitude of warm hugs, and most importantly your unconditional love. Thank you for initiating our first philosophical conversation in the CEED lounge for it has resulted in a beautiful friendship that I could have never imagined. Thank you my sweet Conor. To my kitties Zyra and Kalista, coming home to your meows, messes and purrs will always be

the highlight of my day. Thank you for being the best kitties in the world.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

For centuries, influenza has infected and killed humans. Currently, influenza infects over 1 billion people annually with 3 to 5 million severe cases and a death rate of up to 0.016%, which is approximately 500,000 individuals [14]. There are four types of influenza: A, B C,and D. B and C purely circulate in humans, but A has many compatible hosts. The pervasive and infective subtype influenza A H1N1, commonly known as swine flu, mutates readily [12] and causes more deaths than many of the other types [23]. The 1918 pandemic caused by a novel influenza A H1N1 subtype remains the deadliest known virus of the 20th century. Predicting mutations is critical to developing effective vaccines that minimize the spread of influenza H1N1, thus sustaining healthy individuals, communities, and global societies.

My thesis focuses on exploring the many years of genomic, strain frequency, fitness, and human epidemiology data available for influenza A. I develop a novel approach using reinforcement learning to explore consecutive seasons of influenza and assess observed genetic sequence mutations in the HA protein, which initiates influenza A infectivity. This consecutive seasons approach combines virus genomic data with epidemiological human mortality data. This research utilizes reinforcement learning to explore not yet observed but highly plausible sequence mutations and assess their fitness through incorporating factors related to virus survival and human mortality using Q-learning. The novelty of this approach comes

from the inclusion of environmental factors as fitness features, predicting amino acid sequences as opposed to sequence frequencies, and in representing the problem of sequence prediction in a reinforcement learning framework.

The motivation for this work is saving human lives through developing a machine learning model that effectively predicts mutations in the influenza genome, thus allowing targeted and effective vaccines to be developed and deployed before the annual seasonal flu commences.

1.2 Contributions

Influenza is an eight stranded RNA virus with two glycoproteins: hemagglutinin (HA) and neuraminidase (NA), which enable infection. Every year, the World Health Organization and six collaborating centers meet twice to decide the seasonal vaccine for the northern and southern hemispheres [25]. These vaccines are chosen purely based on HA serology information from observed influenza sequences collected by over 100 national influenza surveillance centers. The HA serology data provides quantitative values on the cross-immunity of a virus, helping choose the virus strain providing the highest immunity against sequences for the season in review.

My goal is to efficiently and effectively predict influenza A H1N1 mutational pathways (that describe the underlying fitness landscape) using a combination of mathematical and machine learning models to assist with the next year's vaccine development. While the HA serology data provides a source of information for vaccine selection, there are many other factors that contribute to a vaccine's efficacy such as host response and virulence and transmissibility determined by other regions of the influenza genome.

My research moves beyond past research that lacked consideration of time [63] or simplified

the mutational space to binary representations [15] or did not account for fitness outside of survival. In fact, I used a new computational problem and work to solve it with a novel reinforcement learning algorithm. I hypothesize that a fitness landscape representing the mutational network of influenza A with a more sophisticated fitness estimate will allow for a better understanding of the potential lethality of future influenza strains with the ultimate goal of providing more effective vaccines to eradicate or mitigate influenza.

Problem Statement Given a data set that contains the protein sequences of the influenza A H1N1 HA gene, vaccine protein sequences, the number of influenza related deaths per month, and the instability index of a given protein sequence, there exists an objective function, $dist(seq_{act}, seq_{pred})$, i.e. the distance between the future predicted and actual sequence, that when minimized leads to the accurate prediction of the future protein sequence.

In this thesis, I make these three contributions to address the above problem statement.

Representing the fitness landscape of mutational pathways. I establish a model to represent the mutational pathways of influenza A H1N1 hemagglutinin (HA) protein strains and its connection to overall fitness. I represent the set of amino acid substitutions leading to the known influenza A strains for H1N1 HA segment as a path in a Markov Decision Process (MDP). When keeping some of the influenza strains hidden based on time precedence from the network, the most likely mutations predicted by the network capture the hidden mutations.

Develop a new fitness model. I develop a fitness model to better capture the effect of influenza A H1N1 HA protein variation, the likelihood of a given strain to result in human death, vaccine similarity, and time sensitivity to the fitness, that is the reproductive rate of a given strain. Building on existing fitness models, the new parameters capture high-level environmental effects in addition to genotypic features.

Predict future observable strains. Utilizing Q-learning, I discover the next season's influenza HA protein for H1N1. Using reinforcement learning with relative fitness as reward, I capture the interactions between sequences and find the optimal path between sequences. This optimal path leads to candidate sequences for the next season's flu.

1.3 Reader's Guide

The thesis consists of multiple chapters, starting with the literature review. Chapter 2 walks through the genetic characteristics of influenza, representing the characteristics, using the characteristics to predict and a description of reinforcement learning. Chapter 3 elaborates on how reinforcement learning is applied to the problem of predicting influenza sequences. Chapter 4 reports the findings from the model runs and Chapter 5 explains the relevance of the findings to the problem objective. Chapter 6 describes the potential next steps to enhance prediction. Appendix A provides an overview of the data available related to influenza prediction and the preprocessing applied to the data used by the model. Appendix B gives a brief explanation of the formulas used for another fitness prediction approach, which can be of use for inverse reinforcement learning, a future work approach. Lastly, Appendix C describes the code structure and provides the raw code for easy adoption.

Chapter 2

Review of Literature

2.1 Genetics of Influenza

Of all types of influenza mentioned in the motivation section 1.1, influenza A has the highest mutation rate [12] and many compatible hosts, suggesting it is the most diverse. Influenza A virus is further classified based on the surface glycoproteins hemagglutinin (HA) and neuraminidase (NA). The glycoproteins are of most importance [46, 75] as they participate in a process called glycosylation, attaching a sugar molecule to a specific amino acid, which modifies the structure of the proteins found in the host. There are 18 and 10 types of HA and NA glycoproteins respectively [25]. Of these combinations, H1N1, H2N2, and H3N2 are the viruses currently circulating in humans [28].

Referring to Bouvier and Palese's paper [7] we can understand the genetics and virus replication cycle of influenza starting with the genome segments. Each influenza A virus subtype contains 8 segments encoding in order: Polymerase acidic sub-units: 1) PB2, 2) PB1, 3) PA, then Hemagglutinin (HA), Nucleoprotein (NP), Neuraminidase (NA), Membrane Protein 1 (M1), and Nonstructural Protein 1 (NS1) proteins [7]. The glycoproteins, HA and NA expressed by segments four and six, initiate and propagate the infection cycle. HA starts the infection cycle as HA0, then cleaves into HA1 which helps bind to the receptor and HA2 which mediates the fusion of the virus envelope with the cell membrane of the host cells. Once bound, NA cleaves terminal sialic acid residues resulting in virus replication and thus

successfully infecting the host cell. Each segment and its role during the virus replication process is depicted in figure 2.1. Given the significance of the HA and NA glycoproteins in virus replication, I focus on the genetics of the glycoprotein encoding segments. Of the glycoproteins, HA is more immunogenic and its genetic characteristics have been more widely researched [7, 44, 66, 80, 81]. Thus, this work primarily focuses on incorporating the genetics of the HA gene, segment four of the influenza A virus.

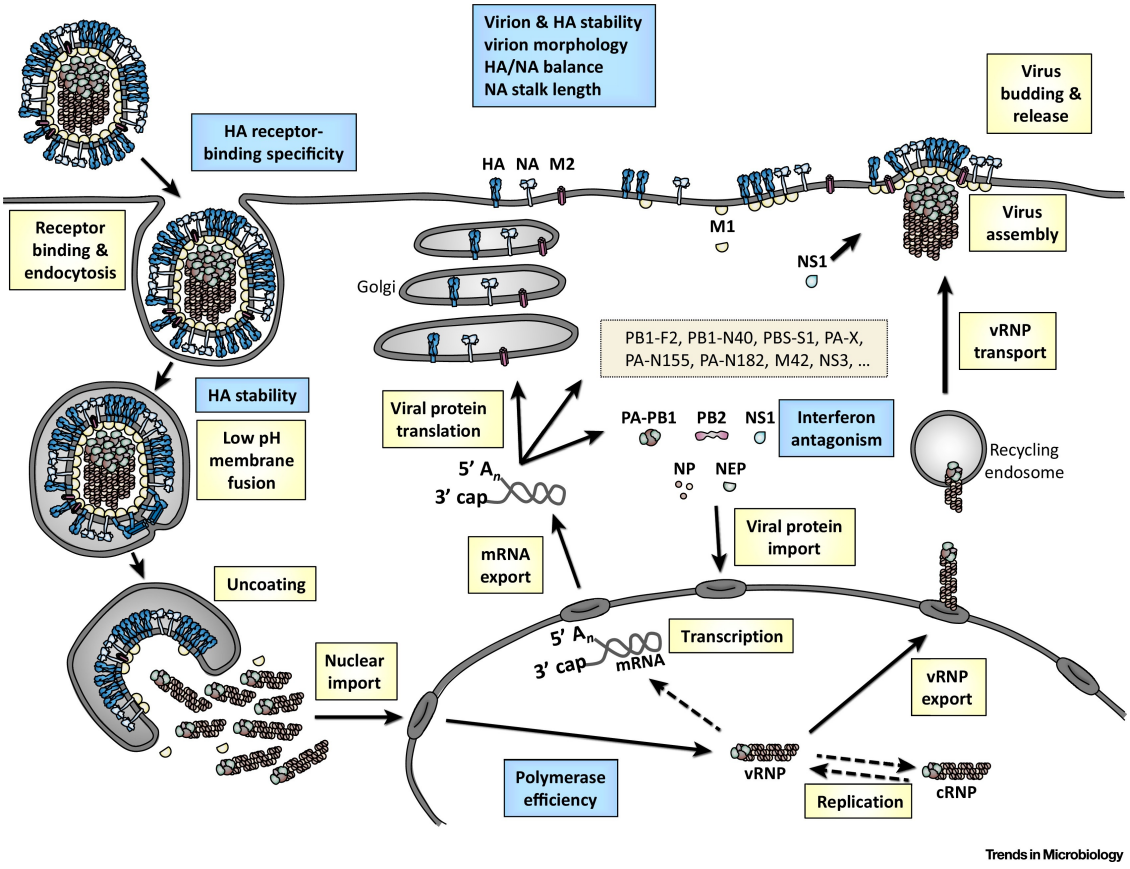


Figure 2.1: The complete cycle of influenza virus replication which leads to flu infection in host cells [66]. Starting at the top left, the HA protein helps bind to the host cell, eventually moving to bottom of the figure depicting the replication of RNA and finally to the top right where the virus is assembled and released.

Virus diversity is driven by antigenic drift and antigenic shift, where drift happens due to mutations of a virus within the same host type and shift happens due to reassortment or

recombination between viruses of same or differing host types [58]. For example, the H3N2 virus is known to have originated from the reassortment between the human H2N2 influenza virus and avian H3N2 influenza virus [74, 78], which is an antigenic shift. As the name suggests, antigenic drift results in small changes, but shifts result in major sections of the genome being altered. Antigenic drifts are more likely to happen than antigenic shifts and thus contribute more to the seasonal influenza infections [25].

Antigenic drift changes occur such that primary protein function is preserved while mutating to enable escaping adaptive immunity [42, 51, 55]. The antigenic regions recognized by the immune system are known as epitopes [1] which are usually the primary sites for antigenic drift. The genetically preserved regions are thus usually in the non-epitopes. There have been many computational studies to define the epitopes within the HA gene [11, 41, 69, 79, 82] and the associated locations highly tolerant to mutations vs. conserved [11, 19].

A key concept within evolution is selection which causes certain mutations to be chosen from the many random mutations that occur. Selection pressure can lead to positive or negative selection, where positive selection (also called Darwinian selection) is the fixation of beneficial variants and negative selection (also called purifying selection) is the purging of deleterious variants [13]. Positive and negative selection have been observed to occur simultaneously in Influenza A H1N1 by separately affecting particular sites within the genome [27]. An interesting result of selection pressure is immune selection, where the influenza virus has grown resistant to antiviral drugs such as oseltamivir through positive selection in NA sites [49]. Although positive selection plays a role in influenza evolution, most amino acid positions evolve under neutral or negative selection [49].

Another genetic characteristic of influenza studied is the change in protein stability. Protein Structure can be denatured by four main methods: application of heat, changes in pH, reaction to chemicals, and reaction to enzymes [4]. Thus, the structural stability of a protein

is dependent on various factors. For thermal stability, Klein et al. have performed correlation analysis of the thermal stability of the HA protein to the Evolutionary Dynamics of influenza [40]. For their data set, the authors considered both experimental and computationally calculated free energy change. They found a correlation between thermal stabilty of HA and the evolutionary dynamics of influenza as measured by phylodynamics, and that the Eris algorithm's computed change in folding free energy matched with the experimental results. For pH stability, Lella et al. reviewed the correlation between pH stability and evolutionary dynamics by analyzing various computational and experimental studies [16]. pH stability plays a major role in HA's ability to mediate fusion during endocytosis. The studies analyzed by Lella et al. revealed structural adaption to different host systems. They also revealed that the presence of histadine residues play a prominent role as pH sensors driving pH stability. Both analysis indicated higher stability meant higher likelihood of antigenic drift and lower stability meant lower likelihood of antigenic drift. The reason being stabilized proteins have the freedom to incorporate destabilizing mutations compared to lower stability proteins.

From this section, it is clear that H1N1 A's HA protein is of importance in replication and thus virus mutants. The epitopes highly tolerant to mutations and protein stability play a key role in which mutations are preferred by the HA protein when replicating.

2.2 Representation of Evolution

2.2.1 Phylogenetic Trees

Phylogenetic Trees are used to characterize and visualize the evolutionary similarity between subpopulations, helping identify orthologous genes. From Kapli et al.'s review on Phylogenetic tree construction published in 2020 [36], we know there are two main phylogenetic tree

reconstruction methods balancing the trade-off between optimality and speed: (1) Sequence-to-sequence Models (character-based) and (2) Distance-based heuristic methods. Sequence-to-sequence Models usually simplify the larger reconstruction problem to a smaller solvable problem by making assumptions such as rooting the ancestor or only considering a particular gene's evolution rather than the complete genome sequence. Common models include Maximum Parsimony and Probabilistic approaches such as Maximum Likelihood and Bayesian. BEAST and the original PhyML are common packages used for Sequence-to-Sequence Models. Distance-based methods include Neighbor Joining (NJ), Minimum Evolution (ME), and Fitch-Margoliash (FM). These distance based methods can handle larger data sets and are computationally faster as only a pair of sequences need to be considered at each iteration, but do not always lead to the optimal solution. Today, most phylogenetic approaches combine the two ideas as seen in FastTree [61], PhyML 3.0 [31] and other variations [45, 53].

In regard to phylogenetic errors, they occur either due to natural sampling bias in the sequence data set or systematic errors due to the violation of the model assumptions as described by Kapli et al. They go on to describe such errors. One such error is Long Branch Attraction (LBA), which means phylogenetic trees tend to favor mapping sequences to be further apart than they are in reality. Another error is that most phylogenetic approaches assume that the substitution process of a species has stayed the same throughout the history of the species. This is called compositional bias. The strategy to tackle some of these systematic errors has been to remove the problematic data points which might result in a technically accurate reconstruction but not always result in a meaningful reconstruction. Kapli et al. continue to mention other systematic errors in phylogenetic trees that need to be researched further. Overall, the likelihood models tend to perform best in lieu of the high computational burden [36].

Within influenza, NextFlu is a well-maintained website which allows scientists to explore

and analyze the evolution of the most recent influenza Virus Sequence [56]. NextFlu focuses on creating a pipeline for processing and visualizing phylogenetic trees, starting by (1) aligning the input sequences, (2) cleaning the aligned data, (3) performing tree reconstruction using FastTree [61] and RAxML [70], (4) inferring internal nodes using marginal maximum likelihood and filling missing leaf nodes with the closest ancestral sequence and (5) adding attributes such as location, time period, frequency trajectory of clades to help visualize and analyze. Figure 2.2 is an example of the influenza A H1N1 sequence trajectory as seen on NextFlu. While NextFlu provides evolutionary information on influenza in an easily consumable form, 2.2, it is meant as a tool for further analysis and is no means the ground truth. Phylogenetic trees in general serve as a starting point for further analysis through understanding Phylodynamics, but cannot be used for prediction of mutational pathways as they do not capture the multidimensional nature of mutations.

2.2.2 Fitness Landscapes

Fitness Landscapes are a concept from Population Genetics introduced by Sewall Wright in 1932. He defines the paradigm fitness landscape as an entire field of gene combinations graded with respect to the adaptive value under a set of conditions [83]. Since then there have been a few different attempts to capture the fitness landscape [2, 26]. Commonly studied fitness landscapes include:

Fisher's geometric model (FGM) [72]: This model was proposed in 1930 and states that mutations that improve the fitness of an organism will be passed on. Fisher's approach was that small mutations were likely to be more successful, such as an single amino acid substitution, and the accumulation of many small mutations improves fitness. Also, simpler phenotypes, ones controlled by a single or a few genes, are more likely to have successful

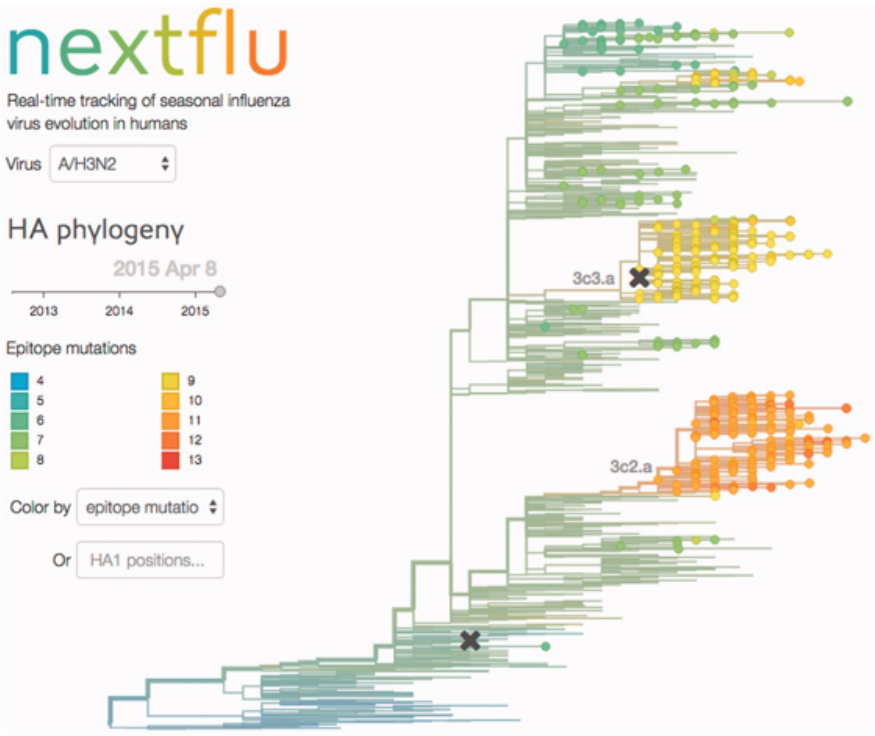


Figure 2.2: NextFlu [56] Phylogenetic Tree displaying USA influenza A H3N2 strain branching. The points marked with X are the sequences chosen for the vaccine composition of the appropriate season.

mutation(s) that could improve fitness. This is known as the "cost of complexity".

Holey Landscapes [29]: This model considers hypervolume areas of high fitness and low fitness. Amino acid sites are either viable (fitness 1) with probability p or lethal (fitness 0) with probability $1 - p$.

House of Cards [30]: This model maintains that mutations can trigger a cascade of changes, essentially reshuffling the genomic "deck", and if the organism survives, then fitness is improved. Thus, any mutation may change the fitness of the protein completely.

Rough Mount Fuji [15]: The Rough Mount Fuji Model builds on the House of Cards model through introducing an (s) factor or additive selective advantage, which can have different values for different loci.

NK Model [37]: A mathematical protein evolution model where k is a key parameter that determines the number of positions at which the amino acid can change simultaneously. This abstract model assumes a sequence of length n with k possible simultaneous mutations and as k increases, the number of steps in the mutation sequence increases while decreasing the size of the steps.

Recently, de Visser et al. [15] published a paper asking and addressing important questions around the predictability and value of fitness landscapes. They bring up the balance between the viability of empirical studies and theoretical models for fitness landscapes. The authors also question the simplistic assumptions made about mutations being strong-selection/weak-mutation (SSWM) and the importance of considering synonymous selections and deleterious mutations. In the end, they leave future researchers with questions including the choice between static and dynamically changing models, choice of topological features, and choice of models to capture the landscape.

While reviewing literature I found Franke et al. [26] and Zagorski et al.'s [86] experimental and theoretical studies on fitness landscape selection insightful. Both papers focus on studying the accessibility of a path from the lowest fitness genotype to the optimal fitness genotype. In particular, Franke et al. answered the following 2 research questions:

1. What is the probability of finding at least one accessible path (represented as $p_L(0)$)
2. What is the expected number of accessible paths? (represented as $< n_L >$)

given a fitness landscape with population N, binary sequences, length of sequence L and

other model specific variables. The authors go on to compare the answers to their research questions for four of the five models described earlier: House of Cards Model which they consider as a null model, Rough Mount Fuji Model, NK model and Holey landscapes. Franke et. al. concludes that the probability of finding at least one accessible path tends to unity as L approaches infinity and the expected number of accessible paths grows as L increases. But, there are a couple of drawbacks in Franke et al.'s analysis of the models; one, they only consider binary sequences and two, they do not account for population dynamics, but do mention plans of doing so in the future. The 1st drawback is addressed and studied by Zagorski et al [86]. In their paper, they perform computational experiments and compare to empirical data sets just as Franke et al. Zagorski et al. demonstrate that as K increases, the number of accessible paths increases, but also becomes longer with increasing number of indirect mutations. The authors also show the dependence of accessibility on the choice of initial fitness. The concepts introduced by the authors that were not discussed by Franke et al. include:

1. Indirect mutations which are either backward mutations, when you decrease in fitness, or distance-neutral mutations, when there is no change in fitness
2. Length of pathways
3. Critical initial fitness above which no accessible path can be found
4. Fitness-distance correlation which refers to nearby genotypes having higher fitness correlation than far away genotypes
5. Predictability of biological evolution when an accessible pathway can cover a significant amount of the fitness landscape.

These are the building blocks when considering any fitness based prediction models.

2.3 Predicting Seasonal Influenza Strain Population

In regard to predicting seasonal influenza strain population, most research has focused on predicting strain frequency using phylodynamics. Two such studies are those of Luksza and Lassig [51] and Huddleston et. al [33]. These are the only two relevant papers as most other papers focus on predicting other aspects of influenza such as whether or not a mutation will take place in a particular position [85], antigenic variants [47, 50], and prediction of antigenic phenotypes [57] as compared to observed influenza strains.

Luksza and Lassig [51] developed a Malthusian fitness model using the influenza virus strains' HA gene segment's epitopic and non-epitopic features. For the epitopic effects, the authors consider the collective interaction of strain frequency and between strain cross-immunity for all substrains in a clade, where strain frequency is defined as the infection rate, cross-immunity is defined as the hamming distance between two strains and a clade is defined as a set of strains that stem from a parent node that has successfully diversified from its ancestors which is inferred from a phylogenetic tree. Being a pioneer in developing a prediction model for influenza strain frequency, the authors provide a solid framework for researchers in computational epidemiology. For next steps, the authors mention incorporating free energy effects on mutations, Haemagglutination Inhibition (HI) Assay data, characteristics of neuraminidase (NA) and geographic properties of strains.

Huddleston et al.'s [33] primary goal is to predict H3N2 strains to increase the accuracy of vaccine strain selection, thus increasing the efficacy of vaccines. Their approach uses Earth Mover's distance as the loss function for a linear additive timeseries fitness model. To cover a range of phenotypic metrics, they divide their data set into: Antigenic Drift metrics - HI antigenic novelty and Epitope Antigenic novelty, Functional constraint metrics - Mutational Load and Deep Mutational Scanning (DMS) mutational effects, and lastly Clade Growth

metrics - Local Branching Index (LBI) and Delta frequency. Using these metrics both individually and in composite models, they evaluated the accuracy of estimated future strains in a simulated environment as well as natural environment. The evaluation included 4 main criteria: (1) Estimated versus observed log_{10} fold change of clade frequency, (2) Percentile rank of the estimated closest strain to fall within 1% and 20% using Hamming Distance, (3) Absolute forecast error, and (4) Spearman correlation between estimated percentile rank and observed percentile rank of future strains. A few conclusions that were made about the metrics being used are: HI antigenetic novelty included forward looking information built into its measurement, models with HI antigenic novelty were more likely to perform better than the naive model and DMS seems to overfit to the strains that were used to perform the DMS experiments. An area of improvement includes incorporating human immune response to the vaccine strains.

A network based approach to map mutational pathways was taken by Wang et al. [76] in 2018. The authors concluded that mutation networks for influenza A H1N1 virus strains between April 2009 to March 2010 are scale-free i.e. the mutations in later time stages are predominantly determined by mutations with higher power in early time stages. To prove this conclusion, the authors first built a mutation network using MJ method in NETWORK software and then found the N_{link}/N_{node} ratio and the diameter of the network. This first step proved that the network was scale free as the ratio only increased in small increments over time i.e. making the mutation types independent of the degree of the graph, and the diameter remained fairly same over time i.e. making mutation types independent of the diameter change. Next they analyze the link-lost when a hub is removed from the network; this is defined as $1 - N_{link,after}/N_{link,before}$. A large link-lost would indicate the particular mutation type to impact future mutations and a small link-lost would indicate no significant impact of that particular mutation type to impact future mutations. In their study, they

found that the link-lost of the most connected hub decreased over time, suggesting it is a predominant mutation type. The authors found such a mutation type in all 8 genes, further establishing that mutations in later time stages are predominantly determined by mutations with higher power in early time stages. They compared the strains of predominant mutation types to the WHO recommendations and found high similarity. They do acknowledge that this study assumes WHO is making the correct vaccine recommendations, but does not address how to circumvent this restriction. They propose that for better understanding of the next best vaccine, one needs to look at the rules governing antigenic drift, viral fitness, the role of source regions, and establishment of predominance. One of the questions that is not addressed by them is how the cycle of year 2009/10 affects their conclusion.

Recently Yin et al. [85] have implemented a time-series mutation prediction model (Tempel) to predict positions that are more likely to mutate within the epitope sites in Hemagluttinin segment of influenza sequences. By using attention-based Recurrent Neural Network, the authors allow for the model to learn the contribution of strains from previous years' to the current year's mutation. From the results for H1N1 except for a couple of test sequences, only the previous year is important for prediction of the current year's mutation locations. Further, the evaluation metrics indicate very little improvement of Tempel when compared to other RNN based approaches. While Tempel achieves high accuracy, the precision and F-score are low, which the authors attribute to the variation in mutation sample size for distinct sites.

2.4 Reinforcement Learning

Within machine learning, there are three main types of learning [71]:

- Supervised: labelled data is used to extrapolate

- Unsupervised: hidden structure of unlabelled data is learned
- Reinforcement Learning: a certain reward is maximized to reach a desired goal

Reinforcement Learning is applied to interactive problems, where a detailed set of true behaviors is unavailable (Supervised learning), and the objective is the outcome rather than the structure of the interactions (Unsupervised Learning). For example, reinforcement learning is often compared to the process of a child learning to walk, where the child is the agent interacting with the uncertain environment,learning the future potential of a successful step through a series of interactions resulting in falls or stable steps. More formally, the mathematical underpinnings of reinforcement learning are defined by Markov Decision Processes (MDP).

2.4.1 Markov Decision Process (MDP)

Definition 2.1. A MDP is a collection of objects $\{T, S, A_s, p_t(\cdot|s,a), r_t(s,a)\}$ [62], where:

- T, a set of decision epochs where each decision epoch, $t \in T$, is a decision made at a certain point of time
- S, a set of possible system states, where a state, $s \in S$ is occupied in the system at each decision epoch
- A_s, a set of all possible actions from state, s
- $p_t(\cdot|s,a)$, the transition probability distribution determining the system state at decision epoch, t upon taking action, a. Thus, the transition probability of going from state, $s \in S$ to $s' \in S$ upon taking action, $a \in A_s$ is denoted as $p_t(s'|s,a)$.
- $r_t(s,a)$, is the reward of taking action, a from system state, s at decision epoch, t.

From a finite-state MDP, one can determine the policy, π, which is a sequence of decision rules taken at a decision epoch; $\pi = \{d_1, d_2, ..., d_t, ...d_N\}$ for N decision epochs. The optimal policy, $\pi*$ is a policy that follows some defined optimal criterion which results in maximum reward for the given environment.

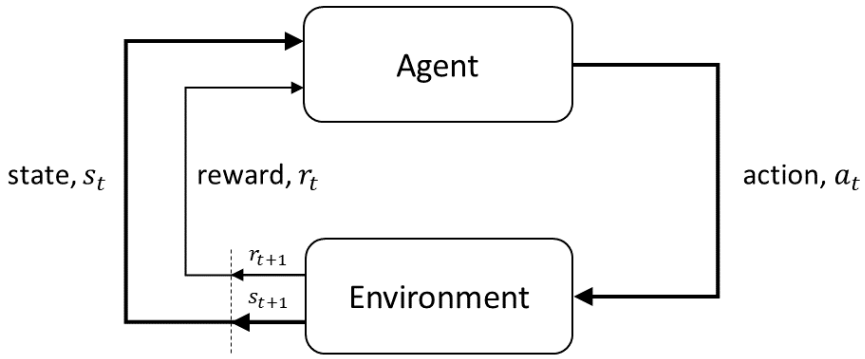


Figure 2.3: Markov Decision Process depicting the interaction between the agent and environment. The Agent receives a state, s_t from the Environment; to which the Agent responds with action a_t receiving a reward of r_{t+1} and moving the Agent to state s_{t+1} [71].

Referring back to the idea of modeling interactions, Figure 2.3 depicts a MDP where the agent interacts with the environment by taking an action, a_t from state, s_t to which the environment responds with an updated state, s_{t+1} and associated reward, $r_t(s_t, a_t)$ for that state-action update. With the example of a child learning in MDP terms, the child is the agent, attempting steps as actions, positions on ground as states and falling or standing as the reward.

Now that the MDP is defined, the challenge becomes solving for an optimal policy, which is performed through the careful selection of the value function.

Definition 2.2. The State-Value Function, $v_\pi(s)$ is the expected return when starting at state, s and following a policy, π. γ is the discount factor for future reward, R_{t+k+1} and G_t is thus the discounted return. k is the number of steps within the current decision epoch, t

[71].

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \forall s \in S \tag{2.1}$$

Definition 2.3. The Action-Value Function, $q_\pi(s, a)$ is the expected return when taking action a starting at state, s and following a policy, π. γ is the discount factor for future reward, G_t is the discounted return, t is the decision epoch and A_t is the set of all possible actions at t [71].

$$q_\pi(s, a) = E_\pi \left[G_t | S_t = s, A_t = a\right] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \tag{2.2}$$

A key phenomenon of value functions is that they are recursive in nature, which is fundamental to the actualization of reinforcement learning.

$$\begin{aligned} v_\pi(s) &= E_\pi \left[G_t | S_t = s\right] \qquad (\text{by } 2.1) \\ &= E_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma E_\pi \left[G_{t+1} | s_{t+1} = s'\right]\right] \\ v_\pi(s) &= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right] \forall s \in S \end{aligned} \tag{2.3}$$

Equation 2.3 is known as the Bellman equation after Richard E. Bellman who came up with the equation in the 1950s in solving the optimal control problem [6].

My goal is to solve for the optimal policy, $\pi*$. To do so, the agent must take the actions that result in $v_{\pi*} \geq v_{pi} \text{for} \pi \in \Pi$. To find such a policy, the Bellman optimality equation must be

applied [71]:

$$
\begin{aligned}
v_*(s) &= \max_{\pi} v_\pi(s) \forall s \in S \\
&= \max_{a \in A(s)} q_{\pi_*}(s,a) \qquad (2.4) \\
&= \max_{a} E_{\pi_*}\left[G_t | S_t = s, A_t = a\right] \\
&= \max_{a} E_{\pi_*}\left[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a\right] \\
&= \max_{a} E_{\pi_*}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right] \\
&= \max_{a} \sum_{s'r} p(s', r|s, a)\left[r + \gamma v_*(s')\right] \qquad (2.5) \\
q_*(s,a) &= E\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \qquad (2.6)
\end{aligned}
$$

Now that I have the Bellman Optimality Equation 2.5 and 2.6 for the state-value function and action-value function, I can look at the different approaches to solving the optimization problem represented by the two equations.

2.4.2 Optimization

In a complete MDP, the state-action transition probability function, $p(s', r|s, a)$ and the associated reward function $r(s', a, s)$ are known. Thus, dynamic programming methods can be used to solve the Bellman Optimality Equation. The two main approaches include:

- **Value Iteration**, where the state-value function for a given state, equation 2.1, can be estimated as the maximum action-value, equations 2.2 and 2.4, over all actions. Iterating through all possible combinations of states and actions until the policy is considered good enough results in the optimal policy. This is a greedy approach, but

has been proven to reach the optimal solution in practice even if the value function has not converged [35, 71].

- **Policy Iteration**, where an arbitrary policy is chosen and iterated over, changing the policy to reflect the best action at each state until the optimal policy is obtained. The best action is chosen by maximizing the state-value function associated with the current policy, equation 2.1. Unlike value iteration, the policy iteration method guarantees the optimal policy if enough iterations are conducted. In practice, the policy iteration method takes pseudopolynomial time [35, 71].

For reinforcement learning, while the problem can be defined as an MDP, the transition probability function is not available but is necessary to use the value iteration and policy iteration methodology above. Thus, to solve for optimal policy in reinforcement learning, there are two possible approaches:

- **Model-free learning**, where the transition probability function is ignored and the focus is on learning the controller driving the agent to the optimal policy

- **Model-based learning**, where the transition probability function is learned first and then applied to learn the controller driving the agent to the optimal policy

Both of these approaches rely on the samples obtained from interacting with the environment. While Model-based learning is more data efficient, there can be a significant bias towards early observations, making them less conducive to problems with very little known interactions [84]. Thus, I focus on model-free learning.

Within model-free learning, these are the methods that have been widely deployed and successful in recent years [9, 71]:

Q-learning and Deep Q-learning (DQN) : Q-learning [77] is based on the idea of value iteration where the q-values are stored in a table and updated every step in an episode using the equation 2.7.

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r(s,a,s') + \gamma \max_{a' \in A} Q(s',a')) \tag{2.7}$$

The Q-learning update is derived from the Bellman equations (2.3, 2.5, 2.6). Q-learning provides a simple yet effective approach to solving for the optimal policy. Q-learning does have its drawback such as high space and computational complexity as a table is used for value storage and there is an update to be calculated every step. This is where Deep Q-Learning plays a vital role. DQN [21] unlike Q-learning does not use a table to store values and does not compute the true q-value update. Instead DQN approximates and learns the q-value through the parameters of a neural network.

One important aspect of naive Q-learning is the balance between exploration and exploitation. One approach to address the balance is called the ϵ-greedy approach. In this approach, one starts at some ϵ corresponding to the probability of exploration, and then slowly decays ϵ by some λ per episode for some number of episodes [71]. By not holding the rate of exploration and exploitation constant for all episodes, the model is allowed to regularize the balance between exploration and exploitation.

Deep Deterministic Policy Gradient (DDPG) [68]: Unlike DQN, DDPG concurrently learns the Q-function and the policy function. DDPG can only be used for continuous action spaces, which is a limitation for some applications. DDPG is advantageous to DQN in two ways: 1) it uses two target networks to learn from rather than starting from scratch every episode, and 2) it uses experience replay, which means learning from all past experiences rather than just the most recent experience.

Asynchronous Advantage Actor-Critic Algorithm (A3C) [54]: Introduced in 2016, A3C addresses the ability of agents to learn from each other using a global asynchronous network of agents and capitalizes on balance between the value function and policy by updating both at each time step. One big difference from Q-learning is that A3C updates a value called advantage instead of the q-value:

$$A(s_t, a_t; \theta, \theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v) \tag{2.8}$$

where k is bounded by the max number of steps per episode, s_t, a_t are the state and action, $V(s_t; \theta_v), r_t$ are the value function and reward and θ_v and θ are the neural network parameters trained for the value function and policy respectively. The advantage function at a high level helps understand the additional reward obtainable by taking action, a_t as compared to the average value for the state, s_t. Overall A3C lowers the time complexity for learning from multiple agents and reduces the overestimation bias of Q-values by using the advantage function.

Trust Region Policy Optimization (TRPO) [67]: TRPO builds on top of Policy Gradient (PG) methods, addressing the drawbacks of PG methods including: vanishing or exploding gradient, low sample efficiency, difficulty in mapping policy to parameter space and stark negative effects on training with large policy changes. Trust Region is a method of locating the optimal point within a set maximum step size rather than taking the step after deciding the direction as in gradient descent. Added to this method is importance sampling which allows the use of old policies in improving the current policy. TRPO's biggest advantage is that it guarantees monotonic improvement.

Two of the above model-free learning approaches are built on Q-learning. Further, given the relatively light literature for my problem as seen in section 2.3, I choose naive Q-learning

as my reinforcement learning approach. Using Q-learning serves as the foundation for my problem that can be molded into any of the above approaches in the future for enhanced performance.

2.4.3 Reward

One thing key to reinforcement learning is defining a good reward function, but this is a challenging task as the reward function relies heavily on the given discipline/domain and is often found by trial and error [71]. Here are some guidelines for the reward function as stated by Sutton et al. [71] and Matignon et al. [52].

- If learning is slow, then try to design a nonsparse reward signal.
- Utilizing the shaping technique, where the reward signal changes as it learns by encountering easy problems at first and then moving to harder more complex problems.
- Applying inverse reinforcement learning, where the goal is to recover the expert's reward signal from the expert's behavior. This method does require strong assumptions about the environment and of feature vectors for which the reward signal is linear.
- The goal of the agent need not be the same goal as the agent's designer as the agent has computational limitations that the agent's designer does not. For example, evolution provides reward signals that are sensitive to observable predictors of evolutionary fitness than the agent's evolutionary fitness itself.
- Improve reward signals via online gradient ascent, where the gradient is of a higher-level objective function.
- Choosing an appropriate initial Q-value, considering, high initial Q-values lead to more exploration, whereas low initial q-values can lead to less exploration, which slows down

the learning and setting the Q-values to be the same as the future observed q-values results in random behavior.

- Cautiously using progress estimators and potential-based shaping

2.5 Support Vector Regression (SVR)

SVR [5] is a commonly used regression technique when there is no simple linear relationship in the two-dimensional space. On a high level SVR represents the feature vector in a higher-dimensional space and identifies linear relationships in that space. The feature vectors are transformed to a higher-dimensional space through the use of kernel functions, $k(x_i, x)$, such as the Radial Basis Function (RBF):

$$k(x_i, x) = \phi(x_i).\phi(x) \tag{2.9}$$

$$rbf(x_i, x) = exp(-\frac{||x_i - x||^2}{\sigma^2}) \tag{2.10}$$

Once converted, the SVR loss function (equation 2.11) is minimized to find the weight vector, w.

$$loss = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}(\xi_i + \xi*_i) \tag{2.11}$$

$$\text{s.t.} \begin{cases} y_i - (w.\phi(x_i)) - b \leq \epsilon + \xi_i \\ (w.\phi(x_i)) + b - y_i \leq \epsilon + \xi*_i \\ \xi_i, \xi*_i \geq 0, i = 1, ..., m \end{cases} \tag{2.12}$$

To choose optimal C, γ (a hyperparameter for RBF kernel), ϵ parameters and appropriate

kernel function, cross-validation [43] is used.

Chapter 3

Methodology

In the case of virus evolution, a virus continuously interacts with their host cells and organisms with the goal of mutating into a fitter variant [18]. Similarly, in reinforcement learning, the focus is on modeling the interactions of an agent within an environment with the goal of discovering the path of highest reward, Section 2.4. The direct parallel between virus evolution and reinforcement learning inspired the use of reinforcement learning to solve for mutational pathways leading to the prediction of future fitter variants. Reinforcement learning, unlike many approaches, provides a holistic view of a goal directed agent taking actions in an uncertain environment [71], just as a virus selects mutations in an ever-changing host cell population. Additionally, unlike Phylogenetic Trees, reinforcement learning allows for deviations from paths of highest reward through the balance of exploration versus exploitation. As explained in section 2.2.1, phylogenetic trees usually assume substitution models to be the same through the seasons, but with a fitness based reinforcement learning model there is more flexibility without too much additional computational burden to include time sensitive features. The reinforcement learning approach builds on the ideas and properties of the fitness landscapes described in section 2.2.2 such as the concept of indirect mutations, the importance of choosing an appropriate initial fitness, and the idea of relative fitness. By learning the mutational paths, we are indirectly learning the underlying fitness landscape. Finally, Lukza and Lassig [51], Huddleston et al. [33], Wang et al. [76] and Yin et al. [85] provide a fitness based growth model, amino acid distance based evaluation metric, the

idea of future mutations depending on high power mutations in the past, and the potential for machine learning methods in predicting evolution respectively. These insights and their respective models' drawbacks act as a guideline for the training and structure of the reinforcement learning model.

The first step to applying reinforcement learning is to define my problem of predicting influenza sequence as a MDP. As described in Section 2.4.1, the attributes of an MDP, as applied to my problem are:

- **States,** S are all possible influenza protein sequences both observed and unobserved. Given that there are $k = 20$ possible amino acids and $n = 566$ possible positions, there are $k^n = 20^{566} = 2.42 * 10^{736}$ possible states.
- **Actions,** A are all possible single point mutations in a given influenza protein sequence. Given that there are $k = 20$ possible amino acids and $n = 566$ possible positions, there are $k * n = 20 * 566 = 11320$ possible mutations/actions for any given sequence, s. Every action is defined as a tuple of the position of mutation and the amino acid to be mutated into.
- **Reward,** $R(s, a)$ is defined as the increase/decrease in fitness of a given sequence, s, upon applying action, a and reaching the future sequence, s'.
- **Transition Probability,** $T(s, a, s')$ is defined as the likelihood of a given sequence, s, upon applying action, a, to move to a future sequence, s'. Q-learning being a model-free approach does not consider learning the transition probability in solving for the optimal policy [71].

For example, mutating sequence state, s = 'MKAILVVLLYTFATSLQCRICI' using action, a = (7, 'M'), results in future sequence state, s' = 'MKAILVVMLYTFATSLQCRICI'. This

change has some reward, $R(s, a) \in \mathbb{R}$. In the next section, I will elaborate on the Reward calculation.

3.1 Reward

In Section 2.4.3, the different considerations while defining the reward function were discussed. The key consideration is that the best reward signal often comes from trial and error, just as the dynamic between mutations and selection as explained in section 2.1. Thus, to start, my reward is defined as the relative change in fitness of the future sequence from the current sequence-based on some known fitness feature vectors. By using fitness as the basis for reward I combine the idea of sequence prediction with learning the underlying fitness landscape.

3.1.1 Fitness function

As mentioned before, "fitness" is a measure of reproductive success of the gene. One way to measure reproductive success is using the observed strain frequency as the fitness. To offset the sampling bias in sequence collection, a ratio of the observed strain frequency and initial strain frequency as obtained from lab experiments is used. This method is explained further in section appendix A. As used by Lukza and Lassig [51] and Huddleston et al. [33], I also use the Malthusian growth model to define the relationship between fitness and strain frequency for observed sequences:

$$P_f(seq) = P_0(seq) * \exp fit(seq) \tag{3.1}$$

where seq is any sequence in the observed state space, P_f is the final frequency or in my case observed frequency, P_0 is the initial frequency or in our case the experimentally obtained frequency, and $fit(seq)$ is the fitness function that is of utmost interest. Rearranging equation 3.1, we get:

$$fit(seq) = \ln \frac{P_f(seq)}{P_0(seq)} \tag{3.2}$$

For sequences that have been observed in nature, both the estimated observed frequency and the experimental frequency are known, but sequences discovered as a part of exploration within the reinforcement learning model, do not have the estimated observed frequencies. Thus, the development of a fitness function such that it can map the fitness feature vectors to the fitness defined in equation 3.2 is needed. There are many fitness feature vectors that can be considered. Historically, sequence-based features' relation to fitness have been studied extensively with little to no inclusion of phenotypic and environmental factors [33, 51, 76]. In this work, fitness is defined not just based on the protein sequence but also the mortality rate and vaccine similarity (termed as cross-immunity by Lukza and Lassig [51]), accounting for environmental factors. Phenotypic information is sparse as described in appendix A and thus not used in the fitness function. Table 3.1, lists all the data used to build the fitness function.

The goal of the fitness function is to capture the mapping between the different fitness feature vectors and the numerical fitness in order to predict the fitness of unknown sequences. To capture the mapping, regression was used, specifically Support Vector Regression (SVR), detailed in section 2.5. The low linear correlation of the feature vectors to the fitness urges to look for a potential linear relationship in a higher dimension between the feature vectors and the fitness, as demonstrated in fig. 3.1. SVR uses kernels to represent the feature vector in a higher dimensional space, which is beneficial to capture the underlying linear relationships. Thus, SVR was trained on the observed influenza data set to predict the

fitness for unknown sequences that are visited in the state space during exploration. The results of the SVR model are discussed in section 4.3.

Note: The mortality rate for an unobserved sequence is set to be the same value as the mortality rate of the sequence most similar to the unobserved sequence. Instead of adding an additional possibility for propagating error by building a prediction model for mortality, I decided to simply base the mortality of unobserved sequences on similarity. Please refer to appendix A for explanation on how mortality was defined for known sequences.

3.1.2 Completion Criteria (also known as Goal State)

At every mutation, the completion criteria is tested, where the mutation is considered to have reached the desired sequence when the future sequence, s' is close to one of the actual future sequence, s_{act} by a distance threshold of θ. The completion criteria , is_done, can be expressed as:

$$is\_done(seq) = 0.0$$

Table 3.1: Overview of the data set used for Fitness Prediction

Feature		**Type**	**Mean, Median, Std**	**Range**
Instability Index		Float, Continuous	10.64, 10.79, 1.71	[5.94, 14.29]
Mortality Ratio		Float, Continuous	5.9e−06, 4.5e−06, 5.9e−06	[3.8e−08, 2.7e−05]
Vaccine Distance	A_New Caledonia_20_1999	Int, Continuous	2.26, 1, 3.06	[0, 20]
	A_Solomon Islands_3_2006		1.85, 1, 1.87	[0, 15]
	A_Brisbane_59_2007		2.26, 1, 3.06	[0, 20]
	A_California_07_2009		1.93, 1, 1.92	[0, 15]
	A_Michigan_45_2015		1.87, 1, 1.88	[0, 15]
	A_Brisbane_02_2018		1.89, 1, 1.88	[0, 15]
Caton Epitopes		Char, Categorical	N/A	All 20 Amino Acids
High Mutational Tolerance Positions		Char, Categorical	N/A	All 20 Amino Acids

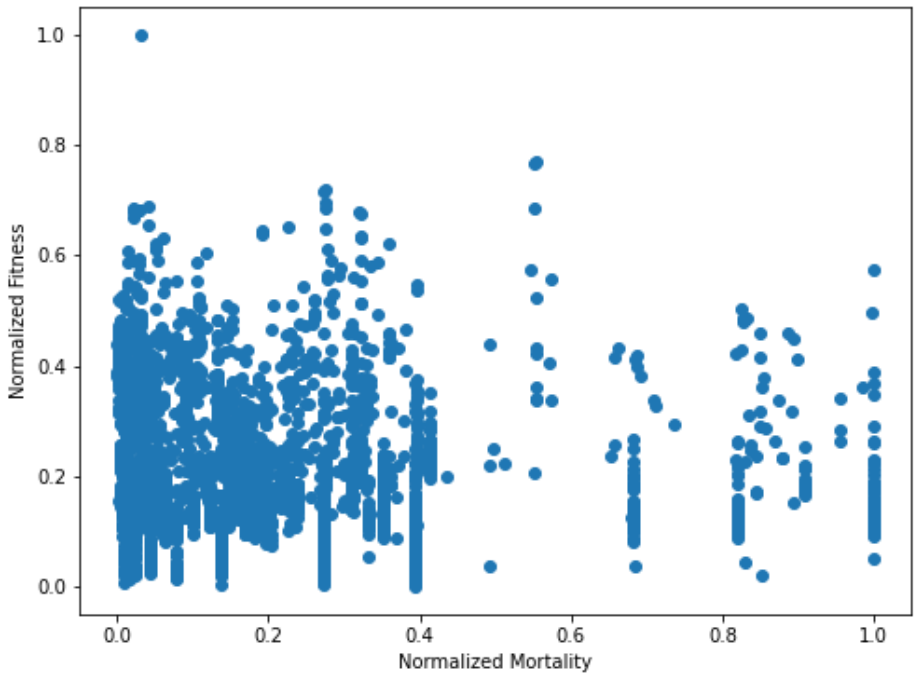


Figure 3.1: The unidentifiable linear relationship between mortality and fitness in the two dimensional space

if $\min_{s_{act} \in S_{act}} dist(s', s_{act}) <= \theta$ **then**

$is\_done(seq) = 1.0$

else

$is\_done(seq) = 1.0/(\min_{s_{act} \in S_{act}} dist(s', s_{act}) - \theta + 1)$

end if

The set, S_{act} is defined as the set of all unique sequences observed in the next season, where current season is the season of the sequence, s. The threshold, θ is the median distance between sequences in S_{act}. The reason for choosing a threshold instead of requiring the predicted sequence to converge to a future sequence is the difficulty of convergence within a reasonable number of steps. If the path of shortest distance is followed, there are d possible paths to get from s to s', where d is the levenshtein distance between s and s'. Levenshtein distance is discussed in eq. (3.5). Since we allow for indirect mutations, where the shortest

path to s' need not be followed and the path of traveling from s to s_i and then back to s is allowed, the number of possible paths to get from s to s' is infinite. Every time a mutation occurs, there are $k^n - 1$ other possible mutations that could have occurred, which means even the smallest deviation from optimal can lead the sequence down a completely different policy than necessary to reach the actual future sequences. Another objective in choosing the completion criteria is to reward the policy when it moves closer to a future sequence and not just when the completion criteria is satisfied. Thus, the completion criteria is defined as a mixture of a threshold and continuous value. Additionally, to reward a policy moving away from a future sequence as zero, the completion criteria is set to be inversely proportional to the distance from the future sequence when it does not meet the threshold. For example, a sequence of distance 10 from the desired sequence will be rewarded $1/10 = 0.1$ as compared to a sequence of distance 1000 which would be rewarded $1/1000 = 0.001$. The completion criteria is only evaluated during training, where the goal is to encourage exploration, allowing for non-optimal policies, while still remaining close to the set of actual future sequences. Lastly, the completion criteria range from $(0, 1]$.

Now, that I have defined the fitness function and completion criteria, I can look at the complete reward equation,

$$r(s, a, s') = fit(s') - fit(s) + \beta_{done} * is\_done(s') \tag{3.3}$$

β_{done} is a regularization parameter, allowing to control the impact of $is\_done(seq)$ value on the reward by scaling it. It is desired to pick a value for β_{done} such that the range for the reward function is still reasonable. Picking a very high or low value will result in a very sparsely distributed reward function. Thus, a number within the fitness value range of $[0, 100]$ was picked, setting the β_{done} value to be 10. Thus, the range of the reward is $(-100, 110]$.

3.2 Training

During the training phase, the main goal is to balance exploration versus exploitation while populating the Q-table. The act of exploring is synonymous to random mutations in evolution and the act of exploiting is synonymous to selection. To balance the exploration versus exploitation, the ϵ-greedy algorithm is used [71] as described in section 2.4.2. For my problem, I start at $\epsilon = 1$, that is, 100% probability of exploration for $min\_explore$ episodes with a decay rate of $\lambda = \frac{\epsilon}{num\_ep//2-1}$ per episode for half the total number of episodes at which point a very small ϵ is then selected, encouraging exploitation for the last half of the episodes. The value of $min\_explore$ is given in section 3.2.1.

The Q-table is initially set to a size of $|S_{known}| \times |A| = 3056 \times 11320$, where $|S_{known}|$ is the cardinality of the unique set of all observed sequences and A is the set of all possible actions. Looping through all the Northern Hemisphere seasons starting with 2008-2009 and ending at 2018-2019, I pair the actual future sequence set, S_{act}, for the given season as the unique set of sequences found in the following season. For example, when in season 2008-2009 or 2012-2013, the set S_{act} is of unique sequences in season, 2009-2010 or 2013-2014 respectively. For each season, I loop through the total number of episodes, $num\_ep_{curr\_season}$, and total number of mutations, $num\_mut_{curr\_season}$, for that season. The starting state for a season is chosen at random following the frequency distribution of that season's sequences and is reset at the end of every episode. Once the starting state is chosen, the q-value is updated according to equation 2.7 for the set number of mutations within the current season or until the completion criteria = 1 is reached. Every time a new state is visited, that is, an unobserved sequence, it is appended to the Q-table, increasing the size of the q-table by one row of the same number of columns. If the complete state-action space is explored, the Q-table will expand to $k^n = 20^{566} = 2.42 * 10^{736}$ rows by 11320 columns which is in-feasibly high in space and computational complexity. Instead I elect to cover the denser regions

of the state-action space. The denser regions capture the high mutational changes which are described by the positions of high mutational tolerance and the amino acids involved in those positions. Positions of high mutational tolerance are defined as those positions which have a max of 93% or less likelihood for any given amino acid in that position. 21 such positions were found in the observed sequence data set. Even within the positions of high mutational tolerance, there are only a few amino acids that are found in that position. To stay within the denser regions, an action, a is chosen from A such that a_{pos} is within the highly mutational positions and a_{amino} is representative of the amino acid distribution for a_pos for the current season. When experimenting, it was observed that the overall amino acid distribution varies drastically from year to year as seen in fig. 3.2. Most notably, the amino acid distribution changes markedly from year 2008 to 2009. To avoid restricting the prediction for next season to only be from the current season's amino acid probabilities, an additional randomness was introduced to the probability, $prob\_rand_{amino}$ defined further in section 3.2.1. While the mutational positions do not change as significantly as the amino acid distributions, I still want to allow for some exploration outside the 21 highly mutational positions. Thus, I introduce a probability of randomness for the positions, $prob\_rand_{pos}$, also defined in section 3.2.1.

The Kullback-Leibler (KL) divergence metric was used to compare the amino acid distribution between seasons. KL is calculated between the season with amino acid distribution, P and another season with the amino acid distribution Q.

$$KL(P||Q) = \sum_i p_i(x) \log(\frac{p_i(x)}{q_i(x)}) \tag{3.4}$$

where i is one of the 20 amino acids, and p_i and q_i are the probability of amino acid i in the respective seasons.

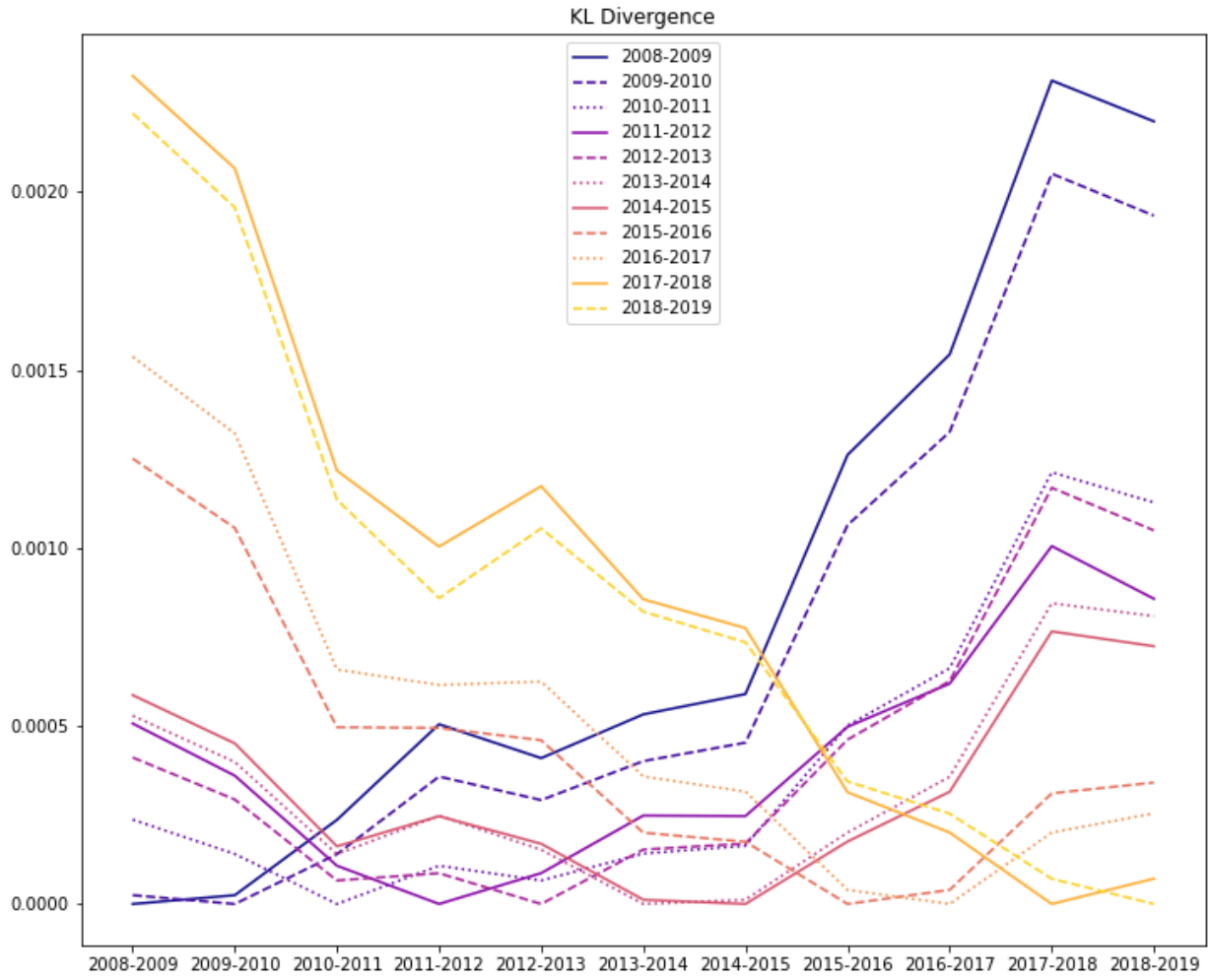


Figure 3.2: Kullback–Leibler (KL) divergence of Amino Acid Distribution between seasons. The color defines a season with amino acid distribution, P and the value on the x-axis defines the amino acid distribution of the season, Q, that P is being compared to.

3.2.1 Hyperparameters

Some of the assumptions made for the hyperparameters and associated setup are derived from the reward function design guidelines. The following considerations were made due to the guidelines in section 2.4.3.

- During the initial runs, I limited my reward function to lie between 0 and 1, but found

the learning to be slow, which I attributed to the Q-values being sparse. To alleviate that problem, I increased the bounds of the reward function to be between -100 and 100 with an extended higher bound of at most 110 for meeting the completion criteria.

- Utilizing the idea of building on easy problems discussed in section 2.4.3, the learning starts with the 2008-2009 season which has a lower number of possible sequences expanding to 2018-2019 which have comparatively high number of possible sequences. The goal is for the model to learn from an easier problem and apply it to more complex problems. Additionally, referring to section 2.3, Wang et al. [76] proves that mutations of higher power from earlier time stages play an important role in mutations in later time stages. Thus, a Q-table that persists across seasons is the ideal approach.

Traditionally, hyperparameters are tuned through multiple iterations over validation sets called cross-validation. When lacking appropriate validation sets, trial and error is the popular solution. Within Q-learning, most studies focus on covering the entire state-action space multiple times [71]. The complete state-action space for my MDP is $2.42 * 10^{736}$ which is infeasible to cover even once. Thus, some of the hyperparameters are chosen based on insight from the observed state-action space. These hyperparameters include:

- **Q-table initialization** (Q_0): To balance the effect of high exploration probability as used in the initial stages of ϵ-greedy approach, the initial values of the Q-table are set to be between $[0, 1]$, which is in the middle of the expected reward range.

- **Number of Episodes** (num_ep): The number of episodes per season is defined by the number of unique sequences for the current season.

- **Number of Mutations within an episode** (num_mut): The number of mutations per season is defined by the number of unique sequences for the next season.

- **Discount Rate** (γ): The discount rate decides the effect of future rewards. For this work, the future states can either be deleterious mutations or advantageous mutations. Population genetics suggest picking more advantageous mutations. With negative reward for deleterious mutations and positive reward for advantageous mutations, the goal is to set the discount rate to a relatively high value that is still less than 1. Thus, $\gamma = 0.90$ was chosen. Additionally, Q_0 values and the discount are inversely proportional [22], thus to balance the mid-range Q_0 values a discount rate of 0.90 was picked.

- **Probability of Exploration** (ϵ): As described earlier in this section, the ϵ-greedy exploration algorithm is used which changes the value of *epsilon* every episode after *min_explore* until half the episodes have been completed.

- **Minimum number of episodes of exploration** (*min_explore*): To ensure the initial Q-values do not drastically affect performance, the model starts by only exploring for the first *min_explore* episodes. This allows the Q-values to be populated with more meaningful values before allowing for exploitation. *min_explore* is set to 0.25 which is 25% of the total episodes with the aim of covering 25% of the total observed state-action space.

- **Probability of choosing Random Position** ($prob\_rand_{pos}$): Most times the actions are chosen from highly mutational positions, but to allow for exploration, some actions are chose completely random positions. $prob\_rand_{pos}$ is the fraction of times a completely random position is chosen.

$$prob\_rand_{pos} = (len(mut\_pos) + len(caton\_epitopes))/566$$

where *mut_pos* and *caton_epitopes* are the positions highly tolerant to mutations as

defined by my data set and by experiments [11] respectively. 566 is the length of all sequences considered in this study. The goal is to choose random positions proportional to ratio of mutational positions.

- **Probability of choosing Random Amino Acid** ($prob\_rand_{amino}$): Most times the chosen action is from the amino acid distribution for a given position, but to allow for exploration, it is desired to choose completely random amino acids sometimes. $prob\_rand_{amino}$ is the fraction of times a completely random amino acid is chosen at a given position.

$$prob\_rand_{amino} = (1 - \min_{prob \forall amino} \max_{prob \forall pos} prob\_amino\_by\_pos)$$

which is the lowest probability majority that an amino acid has for the observed sequence-based probability matrix.

Even covering the observed state-action space is computationally intensive, so 2 other hyperparameters are introduced:

- **Fraction of Episodes to iterate** ($frac\_ep$)
- **Fraction of Mutations to iterate** ($frac\_mut$)

These parameters help reduce the number of episodes and mutations by a defined fraction, making $num\_ep\_frac = frac\_ep * num\_ep$ and $num\_mut\_frac = frac\_mut * num\_mut$. Whenever the number of episodes and number of mutations are mentioned in this work moving forward, it can be assumed that they refer to the fraction of episodes and the fraction of mutations respectively.

To analyze the relationship between the amount of state-action space covered, learning rate,

and sequence predictions, 27 combinations were run such that each of these three hyperparameters: $frac\_ep, frac\_mut$ and α (learning rate) receive one of these three values: $0.25, 0.5$, and 0.75. The observed relationship between the evaluation metrics and the three hyperparameters are detailed in chapter 4.

3.3 Testing

During the testing phase, the Q-table is already populated and the objective is to find the optimal policy by pure exploitation. For the optimal policy, the predictions for each season pair used in training are considered based on the learnt Q-tables for the respective season pair. For example, from the season pair - 2008-2009 and 2009-2010, the pathways most relevant to sequences in 2008-2009 to 2009-2010 are learnt, so now the sequences for 2010-2011 can be predicted by building the optimal policy using the 2008-2009 to 2009-2010 Q-table starting from known sequences in 2009-2010. For each optimal policy, a maximum number of steps, d, is carried out such that $d > 0$ and $d <= 20$. For each unique observed sequence in the next season (in our example, 2009-2010), one gets a sequence prediction, resulting in a distribution of sequence predictions. The evaluation metrics described in the following section 3.3.1 can now be applied to these set of predictions.

3.3.1 Evaluation Metrics

Reward Q-learning is often evaluated by the cumulative or mean reward per episode [60]. Ideally an upward trend in the rewards learned per episode is desired as the model is trained to follow paths of higher return. On the other hand, since the exploration rate is high for the first half of the training, it is not expected that the reward increases from the very first

episode. Of interest is to see if the reward stabilizes after some number of episodes, but is unlikely for my problem as the complete state-action space is not covered.

Policy The learnt policy is the most important result for my problem as it gives insight about the underlying fitness landscape as well as the predicted sequences. As described in the introductory paragraph in the Testing Section 3.3, the optimal policy can be followed to obtain a set of predicted sequences, Seq_{pred} for a given season. For the same season, there are a set of actual sequences, Seq_{act}. One can then compare the two sets against each other using the Levenshtein distance [48], defined as:

$$\mathrm{lev}(seq_{pred}, seq_{act}) = \begin{cases} |seq_{pred}| & \text{if} |seq_{act}| = 0, \\ |seq_{act}| & \text{if} |seq_{pred}| = 0, \\ \mathrm{lev}(\mathrm{tail}(seq_{pred}), \mathrm{tail}(seq_{act})) & \text{if} seq_{pred}[0] = seq_{act}[0] \\ 1 + \min \begin{cases} \mathrm{lev}(\mathrm{tail}(seq_{pred}), seq_{act}) \\ \mathrm{lev}(seq_{pred}, \mathrm{tail}(seq_{act})) \\ \mathrm{lev}(\mathrm{tail}(seq_{pred}), \mathrm{tail}(seq_{pred})) \end{cases} & \text{otherwise.} \end{cases} \tag{3.5}$$

By analyzing the mean, median, and standard deviation of the Levenshtein distance one can understand how close and varied the predictions, Seq_{pred} are from Seq_{act}. Apart from the overall distance, the minimum distance distribution between the best mapped predicted sequence, Seq_{best} to the actual sequence is also analyzed. Seq_{best} is defined as the set of predicted sequences that are closest to the actual sequence creating a one-to-one mapping.

While the median distance gives an idea of how well the model learns all sequences, it does not disclose the prediction accuracy based on strain frequency. To address this, the

weighted average distance between the actual and predicted sequences is calculated. The distance is weighted by the observed frequency for the actual sequence. This metric can only be calculated for the best mapped prediction sequences as there is a one-to-one mapping between the actual predicted sequences.

Additionally, the distance distribution of the sequences within Seq_{pred} are compared to the distance distribution of the sequences within Seq_{act}. This allows to interpret how well the true frequency distribution of sequences is captured. To do so, the Kolmogorov–Smirnov (KS) 2-Sample Test with an $alpha = 0.05$ is applied. A p-value less than α suggests that the two distance distributions are significantly different. For a perfect prediction, the p-value is expected to be greater than 0.05 so that the null hypothesis that the two distributions are the same can be accepted.

Fitness function For the fitness function since regression is used, the traditional evaluation metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R^2 can be applied.

$$\text{MSE} = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2 \tag{3.6}$$

$$\text{MAE} = \frac{1}{n}\sum_{i}^{n}|y_i - \hat{y}_i| \tag{3.7}$$

$$\begin{aligned} R^2 &= 1 - \frac{RSS}{TSS} \\ &= 1 - \frac{\sum_i^n (y_i - \hat{y}_i)^2}{\sum_i^n (y_i - \bar{y})^2} \end{aligned} \tag{3.8}$$

$$\tag{3.9}$$

The closer MSE and MAE are to 0 the better and the closer R^2 is to 1 the better.

Chapter 4

Results

In this section, I explain the results of the 27 Q-learning runs in the Reward and Policy subsections and of the SVR based Fitness Function in the final subsection.

4.1 Reward

Ideally the expectation is that the cumulative reward might initially dip, but eventually increase over episodes to follow a linear line. A positive slope is desired as it indicates an increase in cumulative reward. From table 4.2 and table 4.3, one can notice that the slope is positive for most runs for the seasons: 2008-2009, 2009-2010, 2010-2011, 2011-2012, and 2014-2015. In fact, for seasons: 2013-2014, 2015-2016, 2017-2018, and 2018-2019, the slope is never positive. The hyperparameter sets: 1) $frac\_mut = 0.50, frac\_ep = 0.50, \alpha = 0.25$, 2) $frac\_mut = 0.75, frac\_ep = 0.25, \alpha = 0.75$ and 3) $frac\_mut = 0.75, frac\_ep = 0.50, \alpha = 0.75$ have the highest number of seasons with a positive slope, 7 to be exact. Of these 3 runs, $frac\_mut = 0.75, frac\_ep = 0.25, \alpha = 0.75$ has the highest sum of cumulative rewards over the seasons. To understand the trajectory of the slope better, let us look at the graph of the best run based on reward, that is, the run with $frac\_mut = 0.75, frac\_ep = 0.25, \alpha = 0.75$.

From the graph, figure 4.1, looking at season 2018-2019, there are a series of dips and rises. The rise in reward indicates the agent trying to move towards sequences of higher fitness and the dips indicate the agent exploring sequences of lower fitness. For the seasons with only

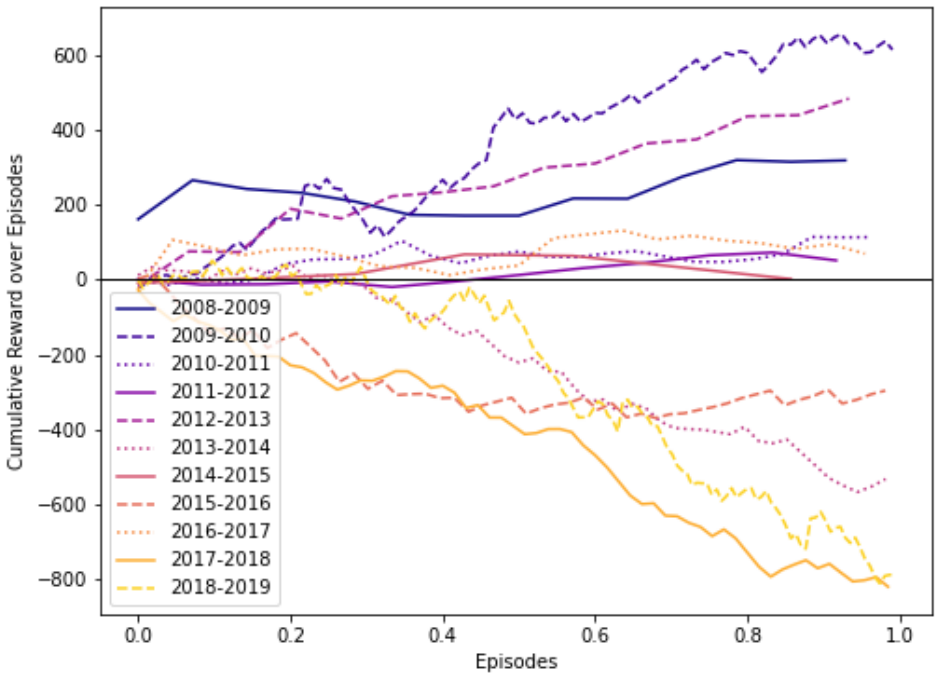


Figure 4.1: Cumulative reward per episode. The x-axis represents the number of episodes scaled to be between 0 and 1 as the number of episodes change per season. The y-axis is the cumulative some of the rewards starting from the first episode until the episode of interest.

negative slope, the explored sequences tend to deviate from the optimal policy enough to not be able get back to the optimal policy. We can see the same from the number of times the completion criteria is met for 2012-2013 which is 8, 2013-2014 which is 54, 2014-2015 which is 6, 2015-2016 which is 53, 2016-2017 which is 20, 2017-2018 which is 65 and 2018-2019 which is 152 for this run as seen in table 4.1. Whenever the completion criteria is met the episode ends and a new start sequence is chosen, learning a new policy. The number of times the completion criteria met is only beneficial for the season 2009-2010.

Table 4.1: Number of times completion criteria is met by season

2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019
9	103	23	11	8	54	6	53	20	65	152

Table 4.2: Part 1 (2008-2009 to 2013-2014): Slope of Cumulative Reward

frac mut	frac ep	α	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014
25	25	25	-121.97	49.96	235.98	78.65	312.66	-490.46
25	25	50	27.87	66.57	194.01	46.70	303.12	-848.71
25	25	75	68.22	-198.59	150.15	-9.32	151.19	-289.18
25	50	25	234.88	882.53	348.74	58.77	263.59	-286.37
25	50	50	-197.46	-262.00	327.18	121.37	382.51	-779.07
25	50	75	-217.87	620.40	291.26	99.42	671.23	-629.52
25	75	25	183.33	1251.86	537.42	47.79	661.36	-1029.47
25	75	50	-15.38	473.87	629.46	101.30	802.84	-639.43
25	75	75	-94.41	712.93	567.18	286.69	669.25	-498.23
50	25	25	92.23	146.39	145.66	-26.57	254.20	-278.61
50	25	50	-103.84	38.69	163.65	63.92	459.86	-564.14
50	25	75	293.05	104.85	288.17	22.78	395.57	-128.81
50	50	25	291.37	247.11	548.55	112.94	831.40	-241.55
50	50	50	420.59	774.19	325.57	171.70	858.71	-781.13
50	50	75	430.65	707.44	443.61	117.99	790.53	-578.15
50	75	25	240.27	1363.19	835.39	-49.41	1186.10	-679.84
50	75	50	426.61	937.37	707.87	218.75	834.89	-726.46
50	75	75	81.95	640.84	658.62	32.21	1234.02	-475.11
75	25	25	173.03	222.15	264.23	103.40	485.26	-272.78
75	25	50	382.91	84.56	235.98	-4.02	335.85	-505.29
75	25	75	115.21	723.47	92.58	95.87	485.55	-679.62
75	50	25	725.90	264.78	366.67	94.75	1070.97	-66.52
75	50	50	423.78	19.08	512.77	174.01	893.41	-579.22
75	50	75	92.80	688.67	212.72	91.54	484.82	-527.28
75	75	25	877.89	830.67	1079.56	260.17	1132.67	-653.32
75	75	50	920.76	740.81	789.26	272.06	1588.83	-427.72
75	75	75	952.45	174.34	928.21	26.95	1186.78	-877.18

4.2 Policy

Looking at table 4.5 and table 4.6, most of the values irrespective of the hyperparameters are the same. On the other hand, the table of best mapped predictions, table 4.7 and table 4.8, is more varied. The median distance of the best mapped predictions to the actual predictions, is the same as the done distance threshold (refer to 'within' row in table 4.4)

Table 4.3: Part 2 (2014-2015 to 2018-2019): Slope of Cumulative Reward

frac mut	frac ep	α	2014-2015	2015-2016	2016-2017	2017-2018	2018-2020
25	25	25	8.53	-837.46	-164.11	-520.05	-663.10
25	25	50	-11.46	-983.79	-175.88	-878.30	-950.36
25	25	75	-31.18	-722.13	-296.57	-389.85	-635.84
25	50	25	3.97	-1780.52	-459.31	-1462.66	-843.55
25	50	50	278.80	-1257.21	-352.00	-1153.74	-1402.02
25	50	75	81.77	-2055.88	-265.06	-1303.31	-1644.85
25	75	25	33.44	-2943.89	-67.28	-1493.33	-2037.67
25	75	50	-115.10	-2353.10	-298.48	-1576.75	-1486.77
25	75	75	59.94	-2816.11	-676.02	-1564.62	-1161.39
50	25	25	-54.41	-901.71	-85.04	-624.79	-678.62
50	25	50	45.38	-1075.50	-262.25	-732.05	-691.76
50	25	75	-128.23	-795.46	-15.62	-1012.45	-796.07
50	50	25	122.62	-1464.54	388.65	-1414.11	-1400.05
50	50	50	166.04	-1355.08	-105.28	-1423.69	-1307.92
50	50	75	91.63	-1587.59	-419.93	-1446.43	-721.73
50	75	25	31.46	-1071.80	-304.02	-1906.49	-1733.27
50	75	50	85.82	-1851.45	-439.08	-2010.90	-2273.77
50	75	75	-55.62	-2178.69	-206.22	-1608.37	-2264.08
75	25	25	-36.10	-734.36	-37.89	-802.51	-934.69
75	25	50	-66.32	-597.36	247.43	-787.32	-1011.75
75	25	75	35.53	-268.01	58.21	-812.90	-900.00
75	50	25	270.59	-1178.31	-630.08	-872.08	-1069.17
75	50	50	-30.08	-1051.19	-233.63	-1091.56	-1490.63
75	50	75	168.37	-938.35	389.67	-1405.96	-1330.97
75	75	25	46.94	-2488.68	-698.79	-2143.16	-1889.42
75	75	50	96.98	-2595.40	-436.31	-2515.91	-2227.76
75	75	75	531.34	-2269.38	-493.75	-2005.40	-2774.57

of the appropriate training start season for most of the seasons. For example, the season 2010-2011's best mapped predictions have a median distance of 5, which is equivalent to the done threshold of 2008-2009 training season. Another point to note is the distance between

the predicted season and its previous season, as the previous season's sequences serve as the starting point for the predicted season. For the 2010-2011 predictions, 2009-2010 sequences are 7 distance away from the 2010-2011 sequences which is greater than 5, the best mapped prediction's median distance for 2010-2011, but less than 8, the overall prediction's median distance for 2010-2011.

Table 4.4: Median distance within and with the next season for actual sequences

	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018
within	5.0	5.0	7.0	8.0	8.0	3.0	4.0	3.0	5.0	6.0
with next	NaN	7.0	10.0	11.0	8.0	4.0	7.0	5.0	9.0	8.0

Additionally, looking at the weighted average of the median distance between the best mapped predicted sequences and the actual sequences, we get table 4.9 and table 4.10. From the table, the run with hyperparameters $frac\_mut = 0.25$, $frac\_ep = 0.50$ and $\alpha = 0.50$ has the overall minimum weighted distance. For that run, the predicted seasons 2013-2014, 2016-2017, 2018-2019 have erroneously high distance values, and the rest of the seasons have a weighted distance within a max of 3 units of the unweighted distance.

The KS test for all 27 runs and for all seasons resulted in p-values less than 0.05. Thus the overall predicted sequence distribution versus the actual sequence distribution as well as the best mapped predicted sequence distribution versus the actual sequence distribution are significantly different from each other.

4.3 Fitness Function

For the SVR Model, I compare my results to a ridge regression model that was optimized for the regularization parameter, λ, using cross validation. The SVR Model to be compared is chosen based on a grid search for hyperparameters: kernel - either 'rbf' or 'linear', C - [0.01,

Table 4.5: Part 1 (2010-2011 to 2014-2015): Median, Mean, Standard Deviation of Distance between all combinations of Predicted and Actual Sequences for the season being predicted

frac mut	frac ep	α	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015
25	25	25	8.0, 8.4, 2.0	11.0, 11.1, 3.5	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 5.0, 1.8
25	25	50	8.0, 8.5, 2.0	11.0, 11.1, 3.6	12.0, 11.7, 3.5	9.0, 9.5, 3.7	5.0, 4.9, 1.8
25	25	75	8.0, 8.4, 2.0	11.0, 11.1, 3.5	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 4.9, 1.8
25	50	25	8.0, 8.5, 2.0	11.0, 11.2, 3.6	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 5.0, 1.8
25	50	50	8.0, 8.4, 2.0	11.0, 11.1, 3.6	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 5.0, 1.8
25	50	75	8.0, 8.4, 2.0	11.0, 11.1, 3.6	12.0, 11.7, 3.4	9.0, 9.4, 3.7	5.0, 5.0, 1.8
25	75	25	8.0, 8.4, 2.0	11.0, 11.2, 3.6	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 4.9, 1.8
25	75	50	8.0, 8.4, 2.0	11.0, 11.1, 3.6	12.0, 11.6, 3.4	9.0, 9.4, 3.7	5.0, 4.9, 1.8
25	75	75	8.0, 8.4, 2.0	11.0, 11.1, 3.6	12.0, 11.7, 3.4	9.0, 9.4, 3.8	5.0, 5.0, 1.8
50	25	25	8.0, 8.4, 2.0	11.0, 11.2, 3.6	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 5.0, 1.8
50	25	50	8.0, 8.4, 2.0	11.0, 11.1, 3.6	12.0, 11.7, 3.4	9.0, 9.4, 3.7	5.0, 4.9, 1.8
50	25	75	8.0, 8.4, 2.0	11.0, 11.1, 3.5	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 4.9, 1.9
50	50	25	8.0, 8.4, 2.0	11.0, 11.1, 3.6	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 5.0, 1.8
50	50	50	8.0, 8.4, 2.0	11.0, 11.2, 3.6	12.0, 11.8, 3.5	9.0, 9.5, 3.7	5.0, 4.9, 1.8
50	50	75	8.0, 8.4, 2.0	11.0, 11.1, 3.6	12.0, 11.7, 3.5	9.0, 9.4, 3.8	5.0, 4.9, 1.8
50	75	25	8.0, 8.4, 2.0	11.0, 11.1, 3.5	12.0, 11.7, 3.4	9.0, 9.4, 3.7	5.0, 4.9, 1.8
50	75	50	8.0, 8.4, 2.0	11.0, 11.1, 3.5	12.0, 11.7, 3.4	9.0, 9.5, 3.7	5.0, 4.9, 1.8
50	75	75	8.0, 8.4, 2.0	11.0, 11.2, 3.5	12.0, 11.7, 3.4	9.0, 9.4, 3.7	5.0, 5.0, 1.8
75	25	25	8.0, 8.4, 2.0	11.0, 11.1, 3.6	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 5.0, 1.8
75	25	50	8.0, 8.4, 2.0	11.0, 11.2, 3.6	12.0, 11.7, 3.4	9.0, 9.4, 3.7	5.0, 4.9, 1.8
75	25	75	8.0, 8.4, 2.0	11.0, 11.1, 3.5	12.0, 11.7, 3.4	9.0, 9.4, 3.7	5.0, 4.9, 1.8
75	50	25	8.0, 8.4, 2.0	11.0, 11.1, 3.5	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 5.0, 1.8
75	50	50	8.0, 8.4, 2.0	11.0, 11.2, 3.6	12.0, 11.7, 3.5	9.0, 9.3, 3.8	5.0, 4.9, 1.8
75	50	75	8.0, 8.4, 2.0	11.0, 11.1, 3.5	12.0, 11.7, 3.5	8.0, 9.3, 3.7	5.0, 5.0, 1.8
75	75	25	8.0, 8.4, 2.0	11.0, 11.2, 3.6	12.0, 11.7, 3.4	9.0, 9.4, 3.7	5.0, 4.9, 1.9
75	75	50	8.0, 8.4, 2.0	11.0, 11.1, 3.5	12.0, 11.7, 3.5	9.0, 9.4, 3.7	5.0, 5.0, 1.8
75	75	75	8.0, 8.4, 2.0	11.0, 11.1, 3.6	12.0, 11.7, 3.4	9.0, 9.4, 3.7	5.0, 5.0, 1.8

0.1, 1, 10], γ - [0.05, 0.5, 5, 50] and ϵ - [0.01, 0.1, 0.2, 0.5, 0.9], using cross-validation. The SVR model with kernel - 'rbf', C - 10, γ - 0.05 and ϵ - 0.1 as the resulting best hyperparameters. The SVR model has a MSE = 0.00996, MAE = 0.08123, and $R^2_{train} = 0.46606$ as compared to Ridgre Regression which has a MSE = 0.01060, MAE = 0.07705 and $R^2_{train} = 0.39021$. The R^2_{test} is comparable for both. Fit of the two approaches is shown in Figure 4.2

Table 4.6: Part 2 (2015-2016 to 2018-2019): Median, Mean, Standard Deviation of Distance between all combinations of Predicted and Actual Sequences for the season being predicted

frac mut	frac ep	α	2015-2016	2016-2017	2017-2018	2018-2019
25	25	25	8.0, 7.3, 1.9	5.0, 6.1, 3.1	10.0, 9.3, 2.5	9.0, 8.7, 2.8
25	25	50	8.0, 7.4, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8
25	25	75	8.0, 7.3, 1.9	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8
25	50	25	7.0, 7.2, 1.9	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8
25	50	50	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8
25	50	75	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8
25	75	25	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.4	9.0, 8.8, 2.8
25	75	50	8.0, 7.3, 1.9	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.7
25	75	75	8.0, 7.3, 1.9	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8
50	25	25	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.4	9.0, 8.7, 2.8
50	25	50	7.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8
50	25	75	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.4, 2.5	9.0, 8.8, 2.8
50	50	25	8.0, 7.3, 1.9	5.0, 6.1, 3.1	10.0, 9.3, 2.5	9.0, 8.8, 2.8
50	50	50	8.0, 7.4, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.4	9.0, 8.8, 2.8
50	50	75	8.0, 7.3, 1.8	6.0, 6.2, 3.1	9.0, 9.3, 2.4	9.0, 8.8, 2.8
50	75	25	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.7
50	75	50	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.4	9.0, 8.8, 2.8
50	75	75	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8
75	25	25	7.0, 7.3, 1.9	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.7, 2.8
75	25	50	8.0, 7.4, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.7
75	25	75	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.7, 2.8
75	50	25	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8
75	50	50	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.4	9.0, 8.7, 2.8
75	50	75	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8
75	75	25	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.7, 2.8
75	75	50	8.0, 7.4, 1.8	5.0, 6.1, 3.1	10.0, 9.4, 2.4	9.0, 8.7, 2.7
75	75	75	8.0, 7.3, 1.8	5.0, 6.1, 3.1	9.0, 9.3, 2.5	9.0, 8.8, 2.8

Note: The regularization parameter, λ used in this subsection is not the same as the hyper-parameter for Q-learning as described in chapter 3.

Table 4.7: Part 1 (2010-2011 to 2014-2015): Median, Mean, Standard Deviation of Distance between best mapped Predicted Sequences to Actual Sequences for the season being predicted

frac mut	frac ep	α	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015
25	25	25	4.0, 4.2, 1.6	5.0, 4.7, 1.7	7.0, 6.2, 2.7	3.0, 2.8, 0.9	3.0, 2.7, 0.9
25	25	50	5.0, 5.3, 2.1	5.0, 5.3, 2.0	6.0, 6.1, 2.3	3.0, 2.9, 0.9	3.0, 2.7, 1.1
25	25	75	5.0, 5.4, 2.2	5.0, 5.1, 1.9	6.0, 5.9, 2.4	3.0, 2.9, 0.9	3.0, 2.7, 1.0
25	50	25	5.0, 5.1, 2.1	5.0, 5.1, 1.9	7.0, 6.5, 2.1	3.0, 2.9, 0.9	3.0, 2.5, 0.8
25	50	50	5.0, 4.7, 1.8	5.0, 5.4, 2.0	7.0, 6.3, 2.6	3.0, 2.9, 1.0	3.0, 2.8, 1.1
25	50	75	5.0, 5.2, 2.2	5.0, 5.4, 1.8	6.0, 6.1, 2.2	3.0, 2.9, 0.9	3.0, 2.6, 1.0
25	75	25	5.0, 5.3, 2.1	5.0, 5.1, 1.9	7.0, 6.3, 2.3	3.0, 2.9, 1.0	3.0, 2.6, 1.0
25	75	50	5.0, 5.0, 1.9	5.0, 5.3, 1.8	6.0, 6.1, 2.4	3.0, 2.8, 0.9	3.0, 2.7, 0.9
25	75	75	5.0, 4.5, 1.6	5.0, 5.0, 1.9	7.0, 6.3, 2.6	3.0, 2.9, 1.0	3.0, 2.6, 0.9
50	25	25	4.0, 4.4, 1.8	5.0, 5.0, 1.9	7.0, 6.5, 2.1	3.0, 2.9, 0.9	3.0, 2.8, 1.1
50	25	50	5.0, 5.1, 2.0	5.0, 5.1, 1.9	6.0, 6.1, 2.3	3.0, 2.8, 0.9	3.0, 2.7, 1.0
50	25	75	5.0, 5.5, 2.2	5.0, 5.4, 1.8	6.0, 5.8, 2.5	3.0, 2.8, 0.9	3.0, 2.7, 1.0
50	50	25	5.0, 4.6, 1.8	5.0, 5.3, 1.8	5.0, 5.9, 2.2	3.0, 2.8, 0.9	3.0, 2.7, 0.9
50	50	50	5.0, 4.8, 1.7	5.0, 4.9, 1.9	6.0, 6.1, 2.3	3.0, 2.9, 1.0	3.0, 2.7, 1.0
50	50	75	5.0, 5.3, 2.2	5.0, 5.0, 1.9	6.0, 6.1, 2.2	3.0, 2.8, 0.9	3.0, 2.6, 0.9
50	75	25	5.0, 5.0, 2.0	5.0, 5.3, 1.8	6.0, 6.2, 2.2	3.0, 2.8, 0.9	3.0, 2.5, 0.8
50	75	50	5.0, 5.0, 1.9	5.0, 5.0, 1.9	5.0, 5.6, 2.5	3.0, 2.8, 0.9	3.0, 2.6, 0.9
50	75	75	5.0, 5.3, 2.1	5.0, 5.0, 1.9	6.0, 5.9, 2.4	3.0, 2.8, 0.9	3.0, 2.8, 1.0
75	25	25	5.0, 4.5, 1.8	5.0, 5.3, 1.8	6.0, 6.1, 2.3	3.0, 2.8, 0.9	3.0, 2.7, 1.0
75	25	50	5.0, 4.7, 1.9	5.0, 5.1, 1.8	6.0, 5.9, 2.4	3.0, 2.8, 0.9	3.0, 2.7, 0.9
75	25	75	5.0, 5.1, 2.1	5.0, 5.3, 1.8	6.0, 6.1, 2.4	3.0, 2.8, 0.9	3.0, 2.6, 0.9
75	50	25	5.0, 4.8, 1.9	4.0, 4.2, 1.6	6.0, 6.2, 2.3	3.0, 2.8, 0.9	3.0, 2.7, 1.0
75	50	50	5.0, 5.0, 2.0	5.0, 4.9, 1.6	6.0, 6.0, 2.4	3.0, 2.9, 1.0	2.0, 2.6, 1.0
75	50	75	5.0, 5.3, 2.2	4.0, 4.4, 1.6	6.0, 5.9, 2.4	3.0, 2.8, 0.9	3.0, 2.5, 0.8
75	75	25	5.0, 5.3, 2.0	5.0, 5.3, 1.8	6.0, 6.1, 2.3	3.0, 2.9, 1.0	3.0, 2.7, 1.0
75	75	50	5.0, 5.1, 1.9	5.0, 5.0, 1.8	7.0, 6.3, 2.6	3.0, 2.8, 0.9	3.0, 2.7, 1.0
75	75	75	5.0, 5.0, 2.5	5.0, 5.0, 2.0	7.0, 6.3, 2.6	3.0, 2.9, 1.0	3.0, 2.8, 1.0

Table 4.8: Part 2 (2015-2016 to 2018-2019): Median, Mean, Standard Deviation of Distance between best mapped Predicted Sequences to Actual Sequences for the season being predicted

frac mut	frac ep	α	2015-2016	2016-2017	2017-2018	2018-2019
25	25	25	2.0, 2.8, 1.5	2.0, 3.0, 1.4	3.0, 3.2, 2.6	5.0, 4.9, 1.8
25	25	50	2.0, 2.7, 1.2	2.0, 2.8, 1.3	3.0, 3.5, 2.4	5.0, 5.0, 1.7
25	25	75	2.0, 2.6, 1.2	2.0, 2.8, 1.4	3.0, 3.6, 2.7	5.0, 4.5, 1.8
25	50	25	2.0, 2.9, 1.4	2.0, 2.3, 1.1	3.0, 3.0, 2.5	5.0, 4.8, 1.8
25	50	50	2.0, 3.0, 1.6	2.0, 2.7, 1.3	3.0, 3.6, 2.9	5.0, 4.7, 1.9
25	50	75	2.0, 2.9, 1.5	2.0, 2.8, 1.4	3.0, 3.3, 1.6	5.0, 4.4, 1.9
25	75	25	2.0, 2.8, 1.5	2.0, 2.8, 1.4	3.0, 3.5, 2.6	5.0, 4.6, 1.8
25	75	50	2.0, 2.6, 1.3	2.0, 2.9, 1.4	3.0, 3.4, 2.4	5.0, 5.0, 1.7
25	75	75	1.0, 2.1, 1.8	3.0, 3.3, 1.7	3.0, 3.3, 2.8	5.0, 4.8, 1.7
50	25	25	2.0, 2.9, 1.5	2.0, 2.5, 1.2	3.0, 3.6, 2.9	5.0, 4.8, 1.8
50	25	50	2.0, 2.8, 1.5	2.0, 2.7, 1.4	3.0, 3.5, 2.7	5.0, 4.4, 1.6
50	25	75	2.0, 2.9, 1.5	2.0, 3.1, 1.5	3.0, 3.5, 2.4	4.0, 4.2, 1.7
50	50	25	2.0, 2.8, 1.2	2.0, 2.4, 1.1	3.0, 3.0, 2.6	5.0, 4.9, 1.8
50	50	50	2.0, 2.8, 1.5	2.0, 3.1, 1.5	3.0, 3.5, 2.4	5.0, 4.9, 1.8
50	50	75	2.0, 2.9, 1.5	2.0, 3.0, 1.5	3.0, 3.5, 2.7	5.0, 4.9, 1.8
50	75	25	2.0, 2.8, 1.5	2.0, 2.9, 1.4	3.0, 3.2, 2.6	5.0, 4.7, 1.8
50	75	50	2.0, 2.9, 1.5	2.0, 2.8, 1.4	3.0, 3.5, 2.4	5.0, 4.5, 1.8
50	75	75	2.0, 2.8, 1.4	2.0, 2.9, 1.5	3.0, 3.5, 2.4	5.0, 4.4, 1.7
75	25	25	2.0, 2.7, 1.3	2.0, 2.8, 1.4	3.0, 3.6, 2.9	5.0, 5.0, 1.8
75	25	50	2.0, 2.8, 1.5	2.0, 3.1, 1.6	3.0, 3.5, 2.4	5.0, 4.4, 1.7
75	25	75	2.0, 2.9, 1.5	3.0, 3.3, 1.6	3.0, 3.5, 2.4	5.0, 4.8, 2.1
75	50	25	2.0, 2.8, 1.5	2.0, 2.7, 1.3	3.0, 3.3, 1.6	5.0, 4.5, 1.8
75	50	50	2.0, 2.8, 1.3	2.0, 2.7, 1.3	3.0, 3.4, 2.4	5.0, 4.5, 1.9
75	50	75	2.0, 2.9, 1.5	2.0, 2.8, 1.4	3.0, 3.3, 1.7	5.0, 4.3, 1.7
75	75	25	2.0, 2.8, 1.5	2.0, 2.4, 1.1	3.0, 3.6, 2.9	3.0, 3.7, 1.6
75	75	50	2.0, 2.7, 1.3	2.0, 2.8, 1.4	3.0, 3.6, 2.9	5.0, 4.5, 1.8
75	75	75	2.0, 2.8, 1.5	2.0, 2.9, 1.4	3.0, 3.3, 2.1	5.0, 4.8, 1.8

Table 4.9: Part 1 (2010-2011 to 2014-2015): Median distance between best mapped predictions and actual sequences weighted by observed sequence frequency

frac mut	frac ep	α	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015
25	25	25	4.08	0.57	3.82	14.98	6.74
25	25	50	8.84	0.64	4.71	15.24	4.80
25	25	75	9.01	0.55	4.84	15.64	5.84
25	50	25	7.74	0.54	7.32	15.71	7.15
25	50	50	5.27	0.63	3.93	15.28	3.57
25	50	75	8.30	0.88	4.71	31.33	5.24
25	75	25	10.06	0.55	7.35	15.14	5.73
25	75	50	6.50	0.57	4.75	15.36	7.64
25	75	75	4.78	0.56	3.87	15.52	4.09
50	25	25	7.03	0.57	7.30	29.91	3.77
50	25	50	7.88	0.57	4.73	15.09	4.21
50	25	75	10.04	0.59	4.69	15.35	9.09
50	50	25	5.11	0.63	5.36	15.00	6.26
50	50	50	6.72	0.59	4.71	15.16	6.21
50	50	75	8.96	0.56	4.65	28.85	9.09
50	75	25	6.37	0.86	5.08	29.64	6.61
50	75	50	6.30	0.54	4.97	15.10	5.97
50	75	75	8.68	0.56	4.80	15.28	6.01
75	25	25	4.77	0.89	4.71	15.16	5.17
75	25	50	4.94	0.58	4.84	29.83	5.88
75	25	75	7.70	0.56	4.59	15.08	7.32
75	50	25	5.46	0.52	4.81	29.71	6.01
75	50	50	7.33	0.59	4.59	15.88	7.22
75	50	75	7.78	0.54	4.80	15.05	4.81
75	75	25	9.38	0.85	4.70	15.67	8.00
75	75	50	7.90	0.57	3.84	29.65	6.13
75	75	75	7.28	0.61	3.94	15.22	3.71

Table 4.10: Part 2 (2015-2016 to 2018-2019): Median distance between best mapped predictions and actual sequences weighted by observed sequence frequency

frac mut	frac ep	α	2015-2016	2016-2017	2017-2018	2018-2019
25	25	25	2.84	30.26	2.57	64.45
25	25	50	2.77	24.96	2.97	75.05
25	25	75	5.02	30.78	3.01	49.51
25	50	25	3.29	21.61	1.96	59.46
25	50	50	3.22	24.96	3.06	44.79
25	50	75	2.98	27.80	4.14	43.32
25	75	25	2.87	25.04	3.16	52.67
25	75	50	5.12	25.80	2.94	65.87
25	75	75	1.41	19.44	2.13	55.46
50	25	25	2.97	33.66	3.06	62.43
50	25	50	2.87	28.52	3.02	33.35
50	25	75	2.96	30.80	2.99	42.74
50	50	25	2.95	24.31	1.95	59.79
50	50	50	2.87	26.70	2.97	58.33
50	50	75	2.97	20.24	3.01	59.33
50	75	25	2.88	35.20	2.55	51.98
50	75	50	2.96	33.57	2.98	38.40
50	75	75	2.88	23.20	2.97	33.65
75	25	25	5.40	27.93	3.06	72.61
75	25	50	2.87	21.55	4.29	41.19
75	25	75	2.97	23.49	4.23	62.78
75	50	25	2.91	28.13	4.78	44.58
75	50	50	2.92	28.46	2.97	38.81
75	50	75	2.89	36.33	2.87	37.77
75	75	25	2.88	23.63	3.06	29.51
75	75	50	2.79	33.17	3.06	38.11
75	75	75	2.88	22.87	3.88	56.34

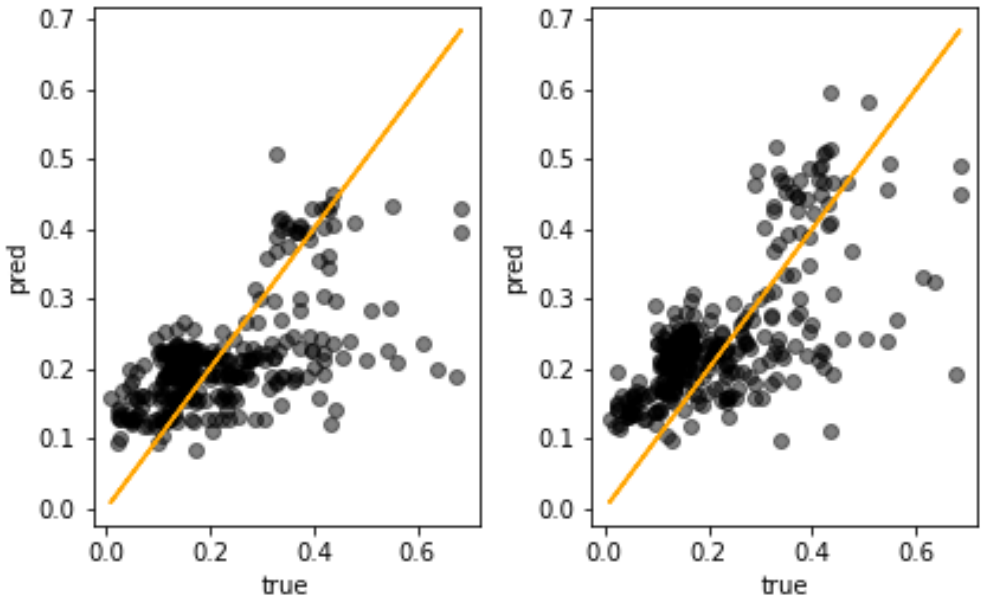


Figure 4.2: Ridge Regression (left) versus SVR (right) Fit. Each point represents the predicted fitness with respect to the true observed fitness.

Chapter 5

Discussion

From the evaluation of Q-learning, the most interesting finding was the stark contrast in learning for seasons before 2013-2014 and seasons after 2013-2014. While we saw an increase in cumulative reward for the former, we saw a decrease for the later. As referred in Section 4.1, the completion criteria is satisfied more often for the seasons with the negative slope compared to the seasons with the positive slope. This can be explained by the distribution of the fitness function and the data samples available.

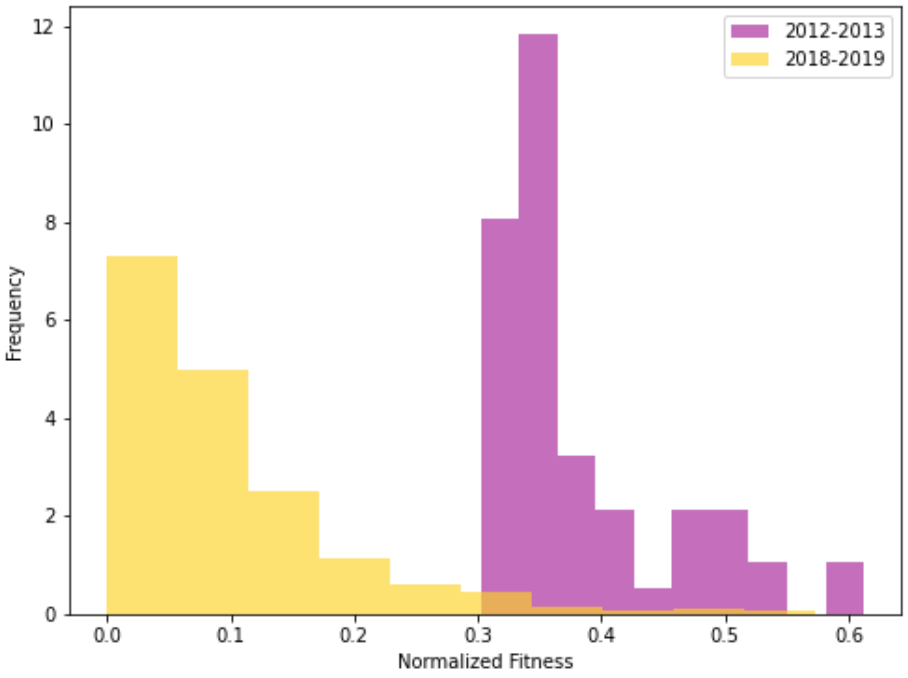


Figure 5.1: Fitness Distribution of Observed Sequences by Season

From the histogram in Figure 5.1, we see that the fitness values are higher and short tailed for 2012-2013 as compared to the fitness values for 2018-2019, which are lower and long tailed. These fitness values are directly proportional to the observed sequence frequency (refer to equation 3.2). With the high rate of meeting the completion criteria for the negative slope seasons, every time the completion criteria is met, a new episode is started which calls for a new starting sequence. If a new sequence is visited often, the likelihood of the new sequence being sampled having a lower fitness is higher as there is a larger distribution of lower fitness sequences for 2018-2019, when compared to 2012-2013. Further, the reason the completion criteria is met many times is the high median distance between sequences for the negative slope seasons. Thus, the stark contrast in learning before and after 2013-2014 can be attributed to sampling bias during data collection of influenza strains.

Following the optimal policy, we get best mapped predictions that mostly lie between the training season's completion criteria threshold and the median distance between the predicted season and preceding season. This suggests that the model is learning the mutational pathways to the extent possible given the constraints of the threshold and sampling bias. Further the weighted average of median distance indicates that the sequences with higher frequency are predicted with the same accuracy as the sequences with lower frequency for most seasons except for the predicted seasons 2013-2014, 2016-2017, and 2018-2019. These seasons have a very skewed distribution of strain frequency with only a few sequences with very high frequencies (refer to figure 5.2). Thus, incorrectly predicting one of these high frequency sequences can sway the weighted distance significantly.

The KS Test results suggest that the model predictions do not cover the actual distribution of sequences. This is expected as the distribution of the sequences per season is quite spread out. For example, season 2017-2018 has sequences with distance between 1 and 22 as seen in fig. 5.3. In the same figure, the predicted sequence distribution all ranges from 1 to 22,

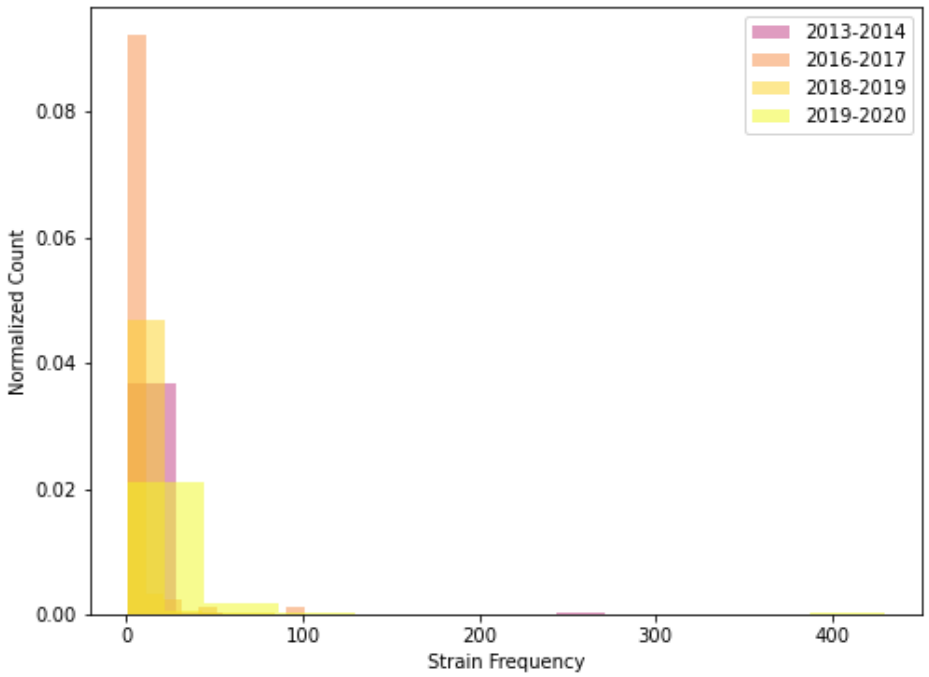


Figure 5.2: Normalized count of the strain frequency for the seasons in the legend

suggesting good coverage of the breadth of the distribution.

Lastly, for the policy features, the median path length for the optimal policy irrespective of the hyperparameters is 2.0, with a max of length of 4.0. The median path length describes the average number of optimal mutations necessary per season to move to a sequence in the following season. This value corresponds well with 2.4, which is the observed average number of new mutations per year for the influenza A H1N1 sequences [10]. Note: the policy is most likely a local optimum and not a global optimum as q-learning does not guarantee convergence with low iterations [71].

Now for the fitness function, as explained in section 3.1.1, the fitness features currently used in the fitness model do not have a very good correlation with the observed fitness values. Given that information it was not expected that for the fitness function to map the fitness features to the estimated fitness values well as observed. Between the SVR and Ridge

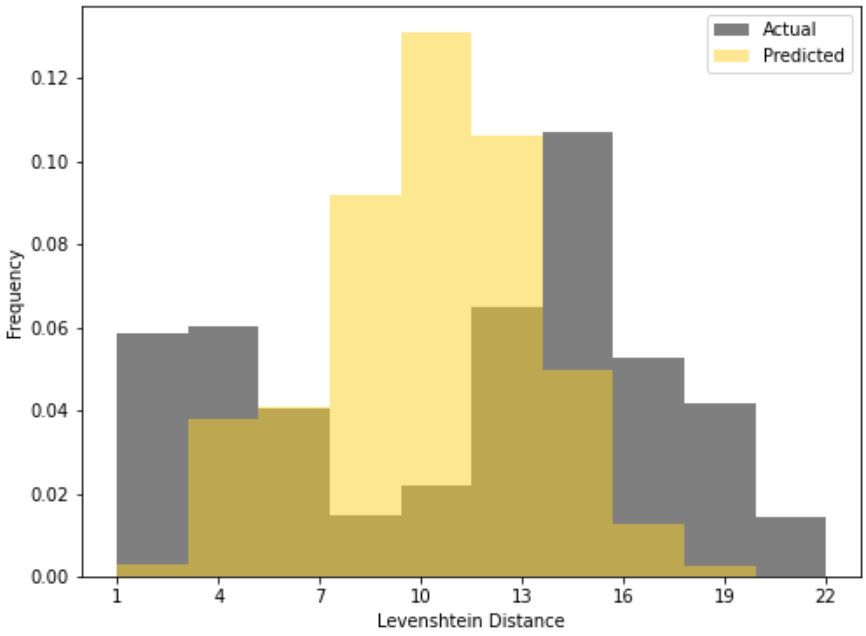


Figure 5.3: Levenshtein Distance between Sequences (Act versus Pred) - 2017-2018

Regression model, SVR was chosen as the spread of the fitness function was better captured as seen in fig. 4.2.

The best model based on the median distance of the best mapped prediction results has an average median distance of 3.6 ± 1.2 between the predicted and actual future sequences. Comparing this to the best result of Huddleston et al.'s model [33], 5.44 ± 1.80, the reinforcement learning model performs better. This statement calls for caution as the reinforcement learning model uses H1N1 HA protein sequences and Huddleston et al.'s model uses H3N2 HA protein sequences. Overall, the reinforcement learning model is able to predict future sequences with reasonable accuracy given the constraints of the parameters and data quality. This demonstrates that the reinforcement learning approach has potential to learn mutational pathways and be a prominent player in guiding vaccine selection with further fine-tuning of the algorithm. There is also potential for this approach to be applied to COVID-19 virus sequence predictions data permitting.

Chapter 6

Conclusions

Applying reinforcement learning to predict future sequences is a novel approach. In fact, previous approaches [33, 51, 76, 85] were not attempting to predict the actual amino acid sequences for the next year from the data known this year, making this work's computational problem a new one. Previous approaches focused on using phylogenetic tree and phylodynamics to predict the frequency of a clade [51] or a sequence [33], or focused on using genetic sequences to classify whether or not a location in the sequence will mutate [85], but did not try to predict the amino acid composition of future sequences. Given this new computational problem and my novel Q-learning approach to influenza strain prediction, there are opportunities to extend my research. I give three below.

The first opportunity is the use of inverse reinforcement learning [59, 64, 88], where given an optimal policy and some known rewards, you can predict the reward function. Such an approach alleviates the need to assign a fitness score to the unobserved sequences. The drawback of this approach is the need for continuous states and a few known mutational pathways. To obtain known mutational pathways, one can use well-established phylogenetic trees and try various mutational pathways that include the key nodes in the tree starting from a parent to a child sequence. Given enough pathway samples, the learned fitness function can be analyzed for accuracy. This approach also allows to parameterize the fitness function as a polynomial, so each fitness factor can have a weight that the algorithm learns. Learning the effect of environmental and phenotypic fitness factors individually and inter-dependently

will help understand the underlying dynamics of the influenza H1N1 virus.

Secondly, there is the possibility to convert from naive Q-learning to Deep Q-learning (DQN), as DQN allows for a larger representation of state-action space by reducing time complexity by approximating expected q-values instead of calculating the expected q-value (refer to Section 2.4 for a description of DQN). Given the large number of possible mutational pathways for influenza virus, it is pertinent to convert to a DQN algorithm.

Thirdly, in terms of data, there is a need for more accurate data obtained directly from the source. Using the concepts of Internet of Things (IOT), infectivity and mortality data should be collected in real time and estimated using advanced network tracing algorithms. Currently, for the environmental factors for unobserved sequences, the value is defined by sequence similarity. Enough advancement has been made in computational epidemiology using SIR models, which can act as a better estimate for unobserved sequences. The recent publication of a newer version of AlphaFold [73] also provides promise for faster protein structure calculations which will indirectly allow the use of more complex and accurate algorithms for protein stability values. Next, to adjust for the current sampling bias in data collection, clustering the sequences based on similarity can be of significance. With phylogenetic tree based predictions of future sequences, the major advantage is aggregated information rather than individual sequence-based information. True improvement in predicting sequences hinges on honing the data quality of the genetic sequences and associated properties. Fludb and GISAID's continued efforts in obtaining metadata for genetic sequences and eliminating sampling bias in data collection will be essential for improved prediction accuracy.

Bibliography

[1] Gillian M Air. Influenza virus antigenicity and broadly neutralizing epitopes. *Current opinion in virology*, 11:113–121, 2015.

[2] Takuyo Aita, Masahiro Iwakura, and Yuzuru Husimi. A cross-section of the fitness landscape of dihydrofolate reductase. *Protein engineering*, 14(9):633–638, 2001.

[3] Rebecca F Alford, Julia Koehler Leman, Brian D Weitzner, Amanda M Duran, Drew C Tilley, Assaf Elazar, and Jeffrey J Gray. An integrated framework advancing membrane protein modeling and design. *PLoS Comput Biol*, 11(9):e1004398, 2015.

[4] Tsutomu Arakawa, Steven J Prestrelski, William C Kenney, and John F Carpenter. Factors affecting short-term and long-term stabilities of proteins. *Advanced drug delivery reviews*, 46(1-3):307–326, 2001.

[5] Mariette Awad and Rahul Khanna. Support vector regression. In *Efficient learning machines*, pages 67–80. Springer, 2015.

[6] E Bellman Richard. Dynamic programming, 1957.

[7] Nicole M Bouvier and Peter Palese. The biology of influenza viruses. *Vaccine*, 26: D49–D53, 2008.

[8] U.S. Census Bureau. Population and housing unit estimates, May 2019. URL https://www.census.gov/programs-surveys/popest.html. [Online; accessed 26-May-2021; retrieved from https://datacommons.org/place/country/USA?topic=Demographics].

[9] Sinan Çalışır and Meltem Kurt Pehlivanoğlu. Model-free reinforcement learning algorithms: A survey. In *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2019.

[10] Robert W Carter and John C Sanford. A new look at an old virus: patterns of mutation accumulation in the human h1n1 influenza virus since 1918. *Theoretical Biology and Medical Modelling*, 9(1):1–19, 2012.

[11] Andrew J Caton, George G Brownlee, Jonathan W Yewdell, and Walter Gerhard. The antigenic structure of the influenza virus a/pr/8/34 hemagglutinin (h1 subtype). *Cell*, 31(2):417–427, 1982.

[12] Rubing Chen and Edward C Holmes. The evolutionary dynamics of human influenza b virus. *Journal of molecular evolution*, 66(6):655, 2008.

[13] Supratim Choudhuri. Chapter 2 - fundamentals of molecular evolution**the opinions expressed in this chapter are the author's own and they do not necessarily reflect the opinions of the fda, the dhhs, or the federal government. In Supratim Choudhuri, editor, *Bioinformatics for Beginners*, pages 27–53. Academic Press, Oxford, 2014. ISBN 978-0-12-410471-6. doi: https://doi.org/10.1016/B978-0-12-410471-6.00002-5. URL https://www.sciencedirect.com/science/article/pii/B9780124104716000025.

[14] Lisa R Clayville. Influenza update: a review of currently available vaccines. *Pharmacy and Therapeutics*, 36(10):659, 2011.

[15] J Arjan GM de Visser, Santiago F Elena, Inês Fragata, and Sebastian Matuszewski. The utility of fitness landscapes and big data for predicting evolution, 2018.

[16] Santiago Di Lella, Andreas Herrmann, and Caroline M Mair. Modulation of the ph

stability of influenza virus hemagglutinin: a host cell adaptation strategy. *Biophysical journal*, 110(11):2293–2301, 2016.

[17] Frank DiMaio, Andrew Leaver-Fay, Phil Bradley, David Baker, and Ingemar André. Modeling symmetric macromolecular structures in rosetta3. *PloS one*, 6(6):e20450, 2011.

[18] Esteban Domingo and Celia Perales. Virus evolution. *eLS*, 2019.

[19] Michael B Doud and Jesse D Bloom. Accurate measurement of the effects of all amino-acid mutations on influenza hemagglutinin. *Viruses*, 8(6):155, 2016.

[20] Stefan Elbe and Gemma Buckland-Merrett. Data, disease and diplomacy: Gisaid's innovative contribution to global health. *Global challenges*, 1(1):33–46, 2017.

[21] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, pages 486–489. PMLR, 2020.

[22] Franklin Cardeñoso Fernandez and Wouter Caarls. Parameters tuning and optimization for reinforcement learning algorithms using evolutionary computing. In *2018 International Conference on Information Systems and Computer Science (INCISCOS)*, pages 301–305. IEEE, 2018.

[23] Centers for Disease Control and Prevention. Cdc wonder, Apr 1995. URL https://wonder.cdc.gov/.

[24] Centers for Disease Control and Prevention. Immunization and respiratory diseases (ncird) home, Apr 2020. URL https://www.cdc.gov/ncird/index.html.

[25] Centers for Disease Control and Prevention, 2021. URL https://www.cdc.gov/. [Online; accessed April-2021].

[26] Jasper Franke, Alexander Klözer, J Arjan GM de Visser, and Joachim Krug. Evolutionary accessibility of mutational pathways. *PLoS computational biology*, 7(8), 2011.

[27] Monica Galiano, Paul-Michael Agapow, Catherine Thompson, Steven Platt, Anthony Underwood, Joanna Ellis, Richard Myers, Jonathan Green, and Maria Zambon. Evolutionary pathways of the pandemic influenza a (h1n1) 2009 in the uk. *PloS one*, 6(8): e23779, 2011.

[28] Steven J Gamblin and John J Skehel. Influenza hemagglutinin and neuraminidase membrane glycoproteins. *Journal of biological chemistry*, 285(37):28403–28409, 2010.

[29] Sergey Gavrilets. Evolution and speciation on holey adaptive landscapes. *Trends in ecology & evolution*, 12(8):307–312, 1997.

[30] John H Gillespie. Substitution processes in molecular evolution. ii. exchangeable models from population genetics. *Evolution*, 48(4):1101–1113, 1994.

[31] Stéphane Guindon, Jean-François Dufayard, Vincent Lefort, Maria Anisimova, Wim Hordijk, and Olivier Gascuel. New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of phyml 3.0. *Systematic biology*, 59 (3):307–321, 2010.

[32] Kunchur Guruprasad, BV Bhasker Reddy, and Madhusudan W Pandit. Correlation between stability of a protein and its dipeptide composition: a novel approach for predicting in vivo stability of a protein from its primary sequence. *Protein Engineering, Design and Selection*, 4(2):155–161, 1990.

[33] John Huddleston, John R Barnes, Thomas Rowe, Xiyan Xu, Rebecca Kondor, David E Wentworth, Lynne Whittaker, Burcu Ermetal, Rodney Stuart Daniels, John W Mc-

Cauley, et al. Integrating genotypes and phenotypes improves long-term forecasts of seasonal influenza a/h3n2 evolution. *Elife*, 9:e60067, 2020.

[34] Francis Crick Institute. WORLDWIDE INFLUENZA CENTRE - WHO CC for Reference & Research on Influenza - THE FRANCIS CRICK INSTITUTE. https://www.crick.ac.uk/sites/default/files/2020-10/Crick_SH2021%20report_Full%20seasonal.pdf, 2020. [Online; accessed 03-May-2021].

[35] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[36] Paschalia Kapli, Ziheng Yang, and Maximilian J Telford. Phylogenetic tree building in the genomic age. *Nature Reviews Genetics*, 21(7):428–444, 2020.

[37] Stuart A Kauffman and Edward D Weinberger. The nk model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of theoretical biology*, 141(2):211–245, 1989.

[38] Elizabeth H Kellogg, Andrew Leaver-Fay, and David Baker. Role of conformational sampling in computing mutation-induced changes in protein structure and stability. *Proteins: Structure, Function, and Bioinformatics*, 79(3):830–838, 2011.

[39] Krishna Praneeth Kilambi and Jeffrey J Gray. Rapid calculation of protein pka values using rosetta. *Biophysical journal*, 103(3):587–595, 2012.

[40] Eili Y Klein, Deena Blumenkrantz, Adrian Serohijos, Eugene Shakhnovich, Jeong-Mo Choi, João V Rodrigues, Brendan D Smith, Andrew P Lane, Andrew Feldman, and Andrew Pekosz. Stability of the influenza virus hemagglutinin protein correlates with evolutionary dynamics. *Msphere*, 3(1), 2018.

[41] Björn F Koel, David F Burke, Theo M Bestebroer, Stefan Van Der Vliet, Gerben CM Zondag, Gaby Vervaet, Eugene Skepner, Nicola S Lewis, Monique IJ Spronken, Colin A Russell, et al. Substitutions near the receptor binding site determine major antigenic change during influenza virus evolution. *Science*, 342(6161):976–979, 2013.

[42] Katia Koelle and David A Rasmussen. The effects of a deleterious mutation load on patterns of influenza a/h3n2's antigenic evolution in humans. *Elife*, 4:e07361, 2015.

[43] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[44] Ivan Kosik, William L Ince, Lauren E Gentles, Andrew J Oler, Martina Kosikova, Matthew Angel, Javier G Magadán, Hang Xie, Christopher B Brooke, and Jonathan W Yewdell. Influenza a virus hemagglutinin glycosylation compensates for antibody escape fitness costs. *PLoS pathogens*, 14(1):e1006796, 2018.

[45] Alexey M Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, and Alexandros Stamatakis. Raxml-ng: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics*, 35(21):4453–4455, 2019.

[46] A Lamb. Paramyxoviridae: the virus and their replication. *Fields virology*, 1996.

[47] Min-Shi Lee and Jack Si-En Chen. Predicting antigenic variants of influenza a/h3n2 viruses. *Emerging infectious diseases*, 10(8):1385, 2004.

[48] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

[49] Wenfu Li, Weifeng Shi, Huijie Qiao, Simon YW Ho, Arong Luo, Yanzhou Zhang, and Chaodong Zhu. Positive selection on hemagglutinin and neuraminidase genes of h1n1 influenza viruses. *Virology journal*, 8(1):1–9, 2011.

[50] Yu-Chieh Liao, Min-Shi Lee, Chin-Yu Ko, and Chao A Hsiung. Bioinformatics models for predicting antigenic variants of influenza a/h3n2 virus. *Bioinformatics*, 24(4):505–512, 2008.

[51] Marta Łuksza and Michael Lässig. A predictive fitness model for influenza. *Nature*, 507 (7490):57–61, 2014.

[52] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. In *International Conference on Artificial Neural Networks*, pages 840–849. Springer, 2006.

[53] Bui Quang Minh, Heiko A Schmidt, Olga Chernomor, Dominik Schrempf, Michael D Woodhams, Arndt Von Haeseler, and Robert Lanfear. Iq-tree 2: new models and efficient methods for phylogenetic inference in the genomic era. *Molecular biology and evolution*, 37(5):1530–1534, 2020.

[54] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[55] Richard A Neher. Genetic draft, selective interference, and population genetics of rapid adaptation. *Annual review of Ecology, evolution, and Systematics*, 44:195–215, 2013.

[56] Richard A Neher and Trevor Bedford. nextflu: Real-time tracking of seasonal influenza virus evolution in humans. *Bioinformatics*, 31(21):3546–3548, 2015.

[57] Richard A Neher, Trevor Bedford, Rodney S Daniels, Colin A Russell, and Boris I Shraiman. Prediction, dynamics, and visualization of antigenic phenotypes of seasonal

influenza viruses. *Proceedings of the National Academy of Sciences*, 113(12):E1701–E1709, 2016.

[58] Martha I Nelson and Edward C Holmes. The evolution of epidemic influenza. *Nature reviews genetics*, 8(3):196–205, 2007.

[59] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[60] David L Poole and Alan K Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.

[61] Morgan N Price, Paramvir S Dehal, and Adam P Arkin. Fasttree 2–approximately maximum-likelihood trees for large alignments. *PloS one*, 5(3):e9490, 2010.

[62] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[63] Lijun Quan, Chengyang Ji, Xiao Ding, Yousong Peng, Mi Liu, Jiya Sun, Taijiao Jiang, and Aiping Wu. Cluster-transition determining sites underlying the antigenic evolution of seasonal influenza viruses. *Molecular biology and evolution*, 36(6):1172–1186, 2019.

[64] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.

[65] Bob Rudis. Retrieve flu season data from the united states centers for disease control and prevention ('cdc') 'fluview' portal [r package cdcfluview version 0.9.4], May 2021. URL https://cran.r-project.org/web/packages/cdcfluview/index.html.

[66] Charles J Russell, Meng Hu, and Faten A Okda. Influenza hemagglutinin protein stability, activation, and pandemic risk. *Trends in microbiology*, 26(10):841–853, 2018.

[67] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[68] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[69] Derek J Smith, Alan S Lapedes, Jan C De Jong, Theo M Bestebroer, Guus F Rimmelzwaan, Albert DME Osterhaus, and Ron AM Fouchier. Mapping the antigenic and genetic evolution of influenza virus. *science*, 305(5682):371–376, 2004.

[70] Alexandros Stamatakis. Raxml version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

[71] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[72] Olivier Tenaillon. The utility of fisher's geometric model in evolutionary genetics. *Annual review of ecology, evolution, and systematics*, 45:179–201, 2014.

[73] Kathryn Tunyasuvunakool, Jonas Adler, Zachary Wu, Tim Green, Michal Zielinski, Augustin Žídek, Alex Bridgland, Andrew Cowie, Clemens Meyer, Agata Laydon, et al. Highly accurate protein structure prediction for the human proteome. *Nature*, pages 1–9, 2021.

[74] Sjouke Van Poucke, Jennifer Doedt, Jan Baumann, Yu Qiu, Tatyana Matrosovich, Hans-Dieter Klenk, Kristien Van Reeth, and Mikhail Matrosovich. Role of substitutions in the hemagglutinin in the emergence of the 1968 pandemic influenza virus. *Journal of virology*, 89(23):12211–12216, 2015.

[75] Ralf Wagner, Thorsten Wolff, Astrid Herwig, Stephan Pleschka, and Hans-Dieter Klenk. Interdependence of hemagglutinin glycosylation and neuraminidase as regulators of influenza virus growth: a study by reverse genetics. *Journal of virology*, 74(14):6316–6323, 2000.

[76] Chengmin Wang, Nan Lyu, Lingling Deng, Jing Wang, Wenwen Gu, Hua Ding, Yan Wu, Jing Luo, Liang Wang, Xueze Lyv, et al. Evolutionary pattern and large-scale architecture of mutation networks of 2009 a (h1n1) influenza a virus. *Frontiers in genetics*, 9:204, 2018.

[77] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

[78] Kim B Westgeest, Colin A Russell, Xudong Lin, Monique IJ Spronken, Theo M Bestebroer, Justin Bahl, Ruud van Beek, Eugene Skepner, Rebecca A Halpin, Jan C de Jong, et al. Genomewide analysis of reassortment and evolution of human influenza a (h3n2) viruses circulating between 1968 and 2011. *Journal of virology*, 88(5):2844–2857, 2014.

[79] DC Wiley, IA Wilson, and JJ Skehel. Structural identification of the antibody-binding sites of hong kong influenza haemagglutinin and their involvement in antigenic variation. *Nature*, 289(5796):373–378, 1981.

[80] Don C Wiley and John J Skehel. The structure and function of the hemagglutinin membrane glycoprotein of influenza virus. *Annual review of biochemistry*, 56(1):365–394, 1987.

[81] Ian A Wilson, John J Skehel, and DC Wiley. Structure of the haemagglutinin membrane glycoprotein of influenza virus at 3 å resolution. *Nature*, 289(5796):366–373, 1981.

[82] Yuri I Wolf, Cecile Viboud, Edward C Holmes, Eugene V Koonin, and David J Lipman. Long intervals of stasis punctuated by bursts of positive selection in the seasonal evolution of influenza a virus. *Biology direct*, 1(1):1–19, 2006.

[83] Sewall Wright. *The roles of mutation, inbreeding, crossbreeding, and selection in evolution*, volume 1. na, 1932.

[84] Fengji Yi, Wenlong Fu, and Huan Liang. Model-based reinforcement learning: A survey. In *Proceedings of the International Conference on Electronic Business (ICEB), Guilin, China*, pages 2–6, 2018.

[85] Rui Yin, Emil Luusua, Jan Dabrowski, Yu Zhang, and Chee Keong Kwoh. Tempel: time-series mutation prediction of influenza a viruses via attention-based recurrent neural networks. *Bioinformatics*, 36(9):2697–2704, 2020.

[86] Marcin Zagorski, Zdzislaw Burda, and Bartlomiej Waclaw. Beyond the hypercube: evolutionary accessibility of fitness landscapes with realistic mutational networks. *PLoS computational biology*, 12(12):e1005218, 2016.

[87] Yun Zhang, Brian D Aevermann, Tavis K Anderson, David F Burke, Gwenaelle Dauphin, Zhiping Gu, Sherry He, Sanjeev Kumar, Christopher N Larsen, Alexandra J Lee, et al. Influenza research database: An integrated bioinformatics resource for influenza virus research. *Nucleic acids research*, 45(D1):D466–D474, 2017.

[88] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Appendices

Appendix A

Data Sets

Sequences Nucleotide sequences are preferred as they capture SNPs better, but due to the large number of nucleotide sequences, the time and space complexity of the problem increases significantly. For the purpose of my model, a protein sequence will help capture the genotype to phenotype relationship just as effectively as using nucleotide sequences. Thus Protein sequences of H1N1 HA gene from 2007 to 2020 found in USA from GISAID [20] were used. Within those 13 years, there are around 15,000 sequences of which 4,500 are unique. To avoid ambiguity in mutational pathways, I exclude any sequences that contain B, Z, X or J which represent aspartic acid or asparagine, glutamic acid or glutamine, unknown amino acid and Leucine or isoleucine respectively. Dropping ambiguous sequences decreased the number of unique sequences to 3,500, but preserved the matching nonambiguous sequences, which accounted for 55% of the dropped sequences. Of the 3,500 sequences, only seasons 2007, 2008, and some of 2009's sequences were of length 565. To keep the problem simple as this is a novel approach, we discard all sequences of length 565, resulting in 537 less unique sequences. To account for the seasonality of influenza, each sequence is a part of a season defined as September to February for the Northern region and March to August for the Southern region. Given all my sequences are observed in the United States, 70% of the sequences fall under the Northern region's season. Finally, I use MAFFT for sequence alignment as recommended by Huddleston et al. [33] to allow for vaccine distance calculation. Only vaccine strains for the seasons between 2006 and 2020 are included in the alignment.

Lastly, 28 sequences which have an average levenshtein distance of 40 or higher from all other sequences were removed and considered junk. In the end there are 3408 unique sequences for the model to use.

Vaccine Distance I start by identifying the vaccine strains for the seasons between 2008-2009 and 2019-2020 using Fludb's listing of WHO Influenza Vaccine Strain Recommendations [87]. Using equation 3.5 I calculate the distance of every vaccine strain used with respect to every unique observed sequence obtained after preprocessing of the sequences. Since the vaccine strain information for each season is only available at the strain name level, I had multiple mappings from the strain name to a sequence. The sequence that had the highest occurrence was chosen to represent the given vaccine strain. Efforts were made to obtain a one to one mapping of the vaccine name to vaccine sequence by conversing with the developers of Fludb, but to no avail.

Phenotypes To incorporate phenotypic characteristics, I refer to the phenotype markers for virulence and transmissibility from the Fludb [87] and Oseltamivir Resistance and Zanamivir Resistance from GISAID's isolate specific metadata [20]. Data on phenotypes is sparse for human host H1N1 viruses with only 22% of sequences having a known response to Zanamivir and Oseltamivir and only 637 sequences identified to have increased virulence and enhanced transmissibility. Further the increased virulence and enhanced transmissibility sequences are not in the same date range as my observed sequences. The drug resistance data is categorical in nature with three possible values: Inconclusive, Sensitive and Resistant. Similarly, I convert the virulence and transmissibility to be categorical with 1 (enhanced virulence or increased transmissibility) and 0 (neither observed). Due to the sparsity and categorical nature of this data, I do not use it in our final fitness model.

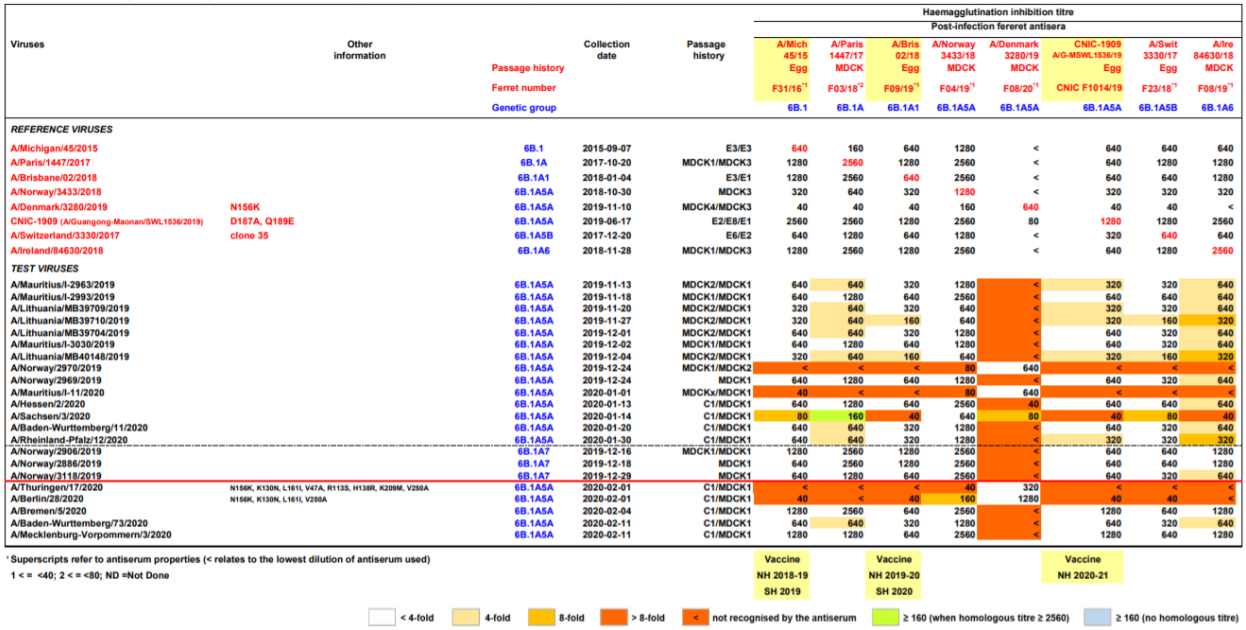
Environmental Factors To understand the evolution of influenza I have to consider the environmental impact of influenza which includes vaccination doses, mortality count by monthly day of the week aggregates and infection rate by week. I obtained all environmental factors from CDC's programs: Wonder [23] for mortality, Vaccine Supply History [24] for vaccine doses, and CDC FluView R package [65] for infection counts. Vaccine doses were only available for the northern region's seasons and Infection counts were for "influenza-like-illness" rather than just influenza. Thus, I decided to only use mortality count for our fitness prediction model. Although mortality is not really related to fitness as compared to infection count, it is the best data available and thus was used as the environmental factor for fitness prediction. To use the mortality data, I first aggregated the day of the week values per month. The next step is to map the mortality count by month to the protein sequences. To do so, I use the collection month from the collection date column provided by the sequence data from GISAID. Since each unique protein sequence can be collected multiple times, I aggregate the mortality for the unique protein sequence by taking a weighted sum of the monthly mortality for every month it occurs. I weight mortality by the number of times the sequence was collected in each month. At the end I converted the mortality counts to mortality rate by dividing the count by the total population for the year. The total population counts were obtained from the US Census [8].

HA Protein Stability Protein Stability calculations tend to take a long time to calculate as they depend on knowing the protein structure. I only know the protein structure of 52 sequences, which means I need to find the protein structure for the rest 3,450 sequences. Given the time complexity of the Rosetta's protein folding algorithm [17] I decided to use Guruprasad et al.'s [32] Instability Index found as a part of the BioPython's SeqUtils.ProtParam module. I want to incorporate RosettaMP's [3, 38, 39] fold energy change calculations based on single mutations in the future. For the Instability Index, any value above 40 is considered

to indicate instability. Since I want to value protein sequences with higher stability,I define $instability_{seq} = 40 - instability\_index_{seq}$. This way a sequence with low stability will have a negative value and vice versa.

Serology Data Hemagluttinin Inhibition (HI) Titer assays are created every season for the northern and southern hemisphere by the World Health Organization's consultation to help select the appropriate vaccine strain. The HA gene of influenza dictates the binding ability of the influenza virus to human Red Blood Cells (RBCs) causing Hemagluttination. Antibodies created by vaccination prevent Hemagluttination or cross-linking of RBCs. The HI Titer assays measure the maximum amount of two-fold dilution of the past season's virus antiserum required to prevent cross-linking of RBCs infected with the current seasonal influenza strains A.1. WHO collects all contributing lab's titer results and creates a clade-based analysis of the titers. For 2019-2020 season of influenza, the observed viruses formed a subclade of 6B.1 creating 6B.1A, which were tested against the reference Michigan virus of clade 6B.1. Unfortunately, the WHO HI Titer data is not publicly available so I tried to use the HI Titer values from the Crick Institute's reports. The raw HI Titer values are part of PDF reports from which I extracted the multiple runs of the Titer assays per season using an online PDF to Excel conversion tool. At this point, the excel files were not well formatted to readily use the titer values. Thus, I decided to focus on other data sources.

Fitness To obtain the initial frequency of the HA influenza sequences, we refer to Doud and Bloom et al.'s work on measuring the effect of amino acid mutations on influenza A H1N1 HA protein [19]. At the end Doud and Bloom arrive at the same conclusion as previous studies on which amino acids in which position are highly tolerant to mutations by producing a preference matrix consisting of each possible amino acid's preference in every given position. The preference matrix is of size $m \times n$, where m is the number of amino acids,



Viruses	Other information		Collection date	Passage history	Haemagglutination inhibition titre: Post-infection fereret antisera							
		Passage history			A/Mich 45/15 Egg	A/Paris 1447/17 MDCK	A/Bris 02/18 Egg	A/Norway 3433/18 MDCK	A/Denmark 3280/19 MDCK	CNIC-1909 A/G-MSWL1536/19 Egg	A/Swit 3330/17 Egg	A/Ire 84630/18 MDCK
		Ferret number			F31/16^{-1}	F03/18^{-2}	F09/19^{-1}	F04/19^{-1}	F08/20^{-1}	CNIC F1014/19	F23/18^{-1}	F08/19^{-1}
		Genetic group			6B.1	6B.1A	6B.1A1	6B.1A5A	6B.1A5A	6B.1A5A	6B.1A5B	6B.1A6
REFERENCE VIRUSES												
A/Michigan/45/2015		6B.1	2015-09-07	E3/E3	640	160	640	1280	<	640	640	640
A/Paris/1447/2017		6B.1A	2017-10-20	MDCK1/MDCK3	1280	2560	1280	2560	<	640	1280	1280
A/Brisbane/02/2018		6B.1A1	2018-01-04	E3/E1	1280	2560	640	2560	<	640	640	1280
A/Norway/3433/2018		6B.1A5A	2018-10-30	MDCK3	320	640	320	1280	<	320	320	320
A/Denmark/3280/2019	N156K	6B.1A5A	2019-11-10	MDCK4/MDCK3	40	40	40	160	640	40	40	<
CNIC-1909 (A/Guangong-Maonan/SWL1536/2019)	D187A, Q189E	6B.1A5A	2019-06-17	E2/E8/E1	2560	2560	1280	2560	80	1280	1280	2560
A/Switzerland/3330/2017	clone 35	6B.1A5B	2017-12-20	E6/E2	640	1280	640	1280	<	320	640	640
A/Ireland/84630/2018		6B.1A6	2018-11-28	MDCK1/MDCK3	1280	2560	1280	2560	<	640	1280	2560
TEST VIRUSES												
A/Mauritius/I-2963/2019		6B.1A5A	2019-11-13	MDCK2/MDCK1	640	640	320	1280	<	320	320	640
A/Mauritius/I-2993/2019		6B.1A5A	2019-11-18	MDCK1/MDCK1	640	1280	640	2560	<	640	640	640
A/Lithuania/MB39709/2019		6B.1A5A	2019-11-20	MDCK2/MDCK1	320	640	320	640	<	320	320	640
A/Lithuania/MB39710/2019		6B.1A5A	2019-11-27	MDCK2/MDCK1	320	640	160	640	<	320	160	320
A/Lithuania/MB39704/2019		6B.1A5A	2019-12-01	MDCK2/MDCK1	640	640	320	1280	<	640	320	640
A/Mauritius/I-3030/2019		6B.1A5A	2019-12-02	MDCK1/MDCK1	640	1280	640	1280	<	640	320	640
A/Lithuania/MB40148/2019		6B.1A5A	2019-12-04	MDCK2/MDCK1	320	640	160	640	<	320	160	320
A/Norway/2970/2019		6B.1A5A	2019-12-24	MDCK1/MDCK2	<	<	<	80	640	<	<	<
A/Norway/2969/2019		6B.1A5A	2019-12-24	MDCK1	640	1280	640	1280	<	640	320	640
A/Mauritius/I-11/2020		6B.1A5A	2020-01-01	MDCKx/MDCK1	40	<	<	80	640	<	<	<
A/Hessen/2/2020		6B.1A5A	2020-01-13	C1/MDCK1	640	1280	640	2560	40	640	640	640
A/Sachsen/3/2020		6B.1A5A	2020-01-14	C1/MDCK1	80	160	40	640	80	40	80	40
A/Baden-Wurttemberg/11/2020		6B.1A5A	2020-01-20	C1/MDCK1	640	640	320	1280	<	640	320	640
A/Rheinland-Pfalz/12/2020		6B.1A5A	2020-01-30	C1/MDCK1	640	640	320	1280	<	320	320	320
A/Norway/2906/2019		6B.1A7	2019-12-16	MDCK1/MDCK1	1280	2560	1280	2560	<	640	640	1280
A/Norway/2886/2019		6B.1A7	2019-12-18	MDCK1	640	2560	1280	2560	<	640	640	1280
A/Norway/3118/2019		6B.1A7	2019-12-29	MDCK1	640	1280	640	2560	<	640	320	640
A/Thuringen/17/2020	N156K, K130N, L161I, V47A, R113S, H138R, K209M, V250A	6B.1A5A	2020-02-01	C1/MDCK1	<	<	<	40	320	<	<	<
A/Berlin/28/2020	N156K, K130N, L161I, V250A	6B.1A5A	2020-02-01	C1/MDCK1	40	<	40	160	1280	40	40	<
A/Bremen/5/2020		6B.1A5A	2020-02-04	C1/MDCK1	1280	2560	640	2560	<	1280	640	1280
A/Baden-Wurttemberg/73/2020		6B.1A5A	2020-02-11	C1/MDCK1	640	640	320	1280	<	640	320	640
A/Mecklenburg-Vorpommern/3/2020		6B.1A5A	2020-02-11	C1/MDCK1	1280	1280	640	2560	<	1280	640	1280
					Vaccine NH 2018-19 SH 2019		Vaccine NH 2019-20 SH 2020			Vaccine NH 2020-21		

*Superscripts refer to antiserum properties (< relates to the lowest dilution of antiserum used)
1 < = <40; 2 < = <80; ND =Not Done

< 4-fold | 4-fold | 8-fold | > 8-fold | < not recognised by the antiserum | ≥ 160 (when homologous titre ≥ 2560) | ≥ 160 (no homologous titre)

Figure A.1: Serology data as represented by the HI Titer assays [34] Each row represents the maximum dilution of the reference virus required to inhibit hemagglutination of RBCs by the test virus.

20, and n is the length of the sequence, 566. Using the experimentally obtained preference matrix, I develop an initial frequency measure for each sequence defined as:

$$P(seq) = \prod_{i=1}^{n-1} p_i(seq[i])$$

$$\log P(seq) = \log \prod_{i=0}^{n-1} p_i(seq[i])$$

$$P_0(seq) = \sum_{i=1}^{n-1} \log_{10} p_i(seq[i]) \tag{A.1}$$

(A.2)

$p_i(aa)$ is the preference of aa at position i. Since the preference values are between 0 and 1, and I am multiplying them to get a preference for a sequence, I take the log of the product. The log preference is then normalized to range between 0 and 1. Note: The first and last

amino acids are start and stop codons, which are always consistent across all influenza A H1N1 HA sequences. Thus, Doud and Bloom do not consider the preference of the first and last amino acids, which I follow.

For the observed (or final) frequency, $P_f(seq)$, we calculate the strain frequency by season and use the derived ratio, that is,

$$P_f(seq) = \frac{\text{count of strain within season}}{\text{total strains for season}} \tag{A.3}$$

P_f is also normalized to range between 0 and 1 so that P_f and P_0 are on the same scale.

Thus,

$$\begin{aligned} fit(seq) &= \ln \frac{P_f(seq)}{P_0(seq)} \qquad \text{by (3.2)} \\ fit(seq) &= \ln \frac{norm(\frac{\text{count of strain within season}}{\text{total strains for season}})}{norm(\sum_{i=1}^{n-1} \log_{10} p_i(seq[i]))} \end{aligned} \tag{A.4}$$

Appendix B

Fitness Function

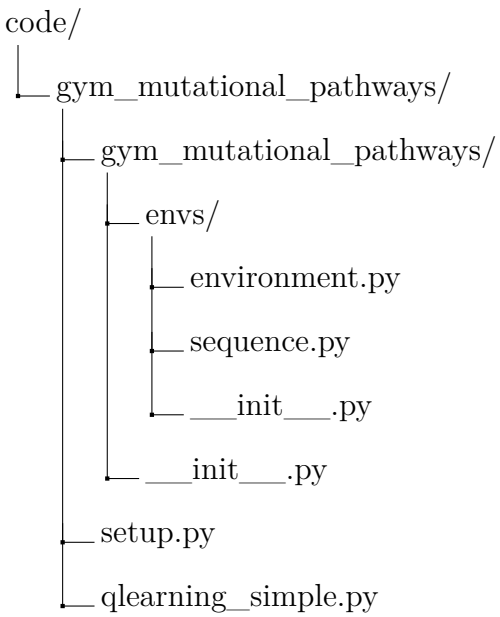
Another approach to define the fitness function is using the complete data set and self-defining weights that can be easily manipulated. This was my first approach. I have five main components to the fitness function: mortality, infectivity, vaccine effect (also defined as cross immunity in Lukza and Lassig's paper [51]), protein stability and drug resistance. One important factor that could not be considered due to time constraints is the HI Titer serology data.

$$
\begin{aligned}
\textbf{fitness\_mortality} &= \frac{\text{strain\_freq} * \text{mortality}}{\text{total\_strains\_per\_year}} \\
\textbf{vaccine\_distance} &= \sum_{\text{vaccine\_strain}} \text{hamming\_distance}(\text{strain}, \text{vaccine\_strain}) \\
\textbf{fitness\_vaccine} &= \text{vaccine\_dist} * \text{vaccine\_dose} \\
\textbf{fitness\_stability} &= 40.0 - \text{instability\_index} \\
\textbf{fitness\_drug\_resistance} &= \text{osel\_resist} * 10.0 + \text{zana\_resist} * 10.0 \\
\textbf{fitness\_total} &= \text{fitness\_stability} + \text{fitness\_vaccine} \\
&\quad + \text{fitness\_mortality} + \text{fitness\_drugs}
\end{aligned} \tag{B.1}
$$

Appendix C

Code

The Code for Q-learning is divided in multiple files starting with *readData.py* for reading the data from different sources and preprocessing. Then a gym AI environment is setup with the following file structure:



The qlearning_simple.py file (C.1) contains the main code for learning and prediction, the environment.py file (C.2) defines the gym AI custom environment for our problem, the sequence.py file (C.3) defines a sequence object used to represent each state and both the ___init___.py (C.4, C.5) and setup.py (C.6) files help with initializing the gym AI custom environment.

```
1 # Sample Command: 'python qlearning_simple.py -datapath "c:\\users\\aarat\\
```

```
    documents\\masters\\research\\" --FRAC_ITERS_MUT 0.5 --FRAC_ITERS_EP 0.5
    --FRAC_MIN_RANDOM 0.25 --THRES_DONE 10 --LEARNING_RATE 0.5 --DISCOUNT 0.75
     --EXPLORE_PROB 0.2 --POSITION_RANDOM_PROB 0.2 --AA_RANDOM_PROB 0.4'

from datetime import datetime as dt
time_taken_ls = []
start_time = dt.now()

## If Error: Cannot re-register id: MutationSpace-v0 run below
import gym
env_dict = gym.envs.registration.registry.env_specs.copy()
for env in env_dict:
    if 'MutationSpace-v0' in env:
        print("Remove {} from registry".format(env))
        del gym.envs.registration.registry.env_specs[env]

import sys
sys.path.append('/home/raarathi/code/')

from scipy.stats import ks_2samp
import numpy as np
import pandas as pd
import gym_mutational_pathways
from gym_mutational_pathways.envs.sequence import Sequence
import matplotlib.pyplot as plt
import seaborn as sns
import random
import os
import sys
import argparse
parser = argparse.ArgumentParser()
```

```
30 parser.add_argument("-datapath", help="full path to data directory", type=str)
31 parser.add_argument('--FRAC_ITERS_MUT', help="fraction of mutations", type=
       float)
32 parser.add_argument('--FRAC_ITERS_EP', help="fraction of episodes", type=float
       )
33 parser.add_argument('--FRAC_MIN_RANDOM', help="fraction of min_random", type=
       float)
34 parser.add_argument('--THRES_DONE', help="levenshtein distance at which
       considered done", type=float)
35 parser.add_argument('--LEARNING_RATE', help="rate of past vs. current reward",
        type=float)
36 parser.add_argument('--DISCOUNT', help="discount of future reward", type=float
       )
37 parser.add_argument('--EXPLORE_PROB', help="probability of exploring vs.
       exploiting (often represented as epsilon)", type=float)
38 parser.add_argument('--POSITION_RANDOM_PROB', help="probability of random
       position", type=float)
39 parser.add_argument('--AA_RANDOM_PROB', help="probability of random amino acid
       ", type=float)
40 parser.add_argument('--datetime', help="datetime of q-table to load", type=str
       )
41
42 parser = parser.parse_args(sys.argv[1:])
43
44 FRAC_ITERS_MUT = 0.25 if parser.FRAC_ITERS_MUT is None else parser.
       FRAC_ITERS_MUT
45 FRAC_ITERS_EP = 0.25 if parser.FRAC_ITERS_EP is None else parser.FRAC_ITERS_EP
46 FRAC_MIN_RANDOM = 0.25 if parser.FRAC_MIN_RANDOM is None else parser.
       FRAC_MIN_RANDOM
47 LEARNING_RATE = 0.3 if parser.LEARNING_RATE is None else parser.LEARNING_RATE
48 DISCOUNT = 0.90 if parser.DISCOUNT is None else parser.DISCOUNT # how
```

```
    important are future actions vs. current action
49
50 time_taken_ls.append(dt.now()-start_time)
51 start_time = dt.now()
52
53 print(f'Loading data...\n{parser.datapath}')
54 datapath = parser.datapath if parser.datapath is not None else 'c:\\users\\
    aarat\\documents\\masters\\research\\'
55 os.chdir(datapath)
56 sys.path.append(datapath)
57 print(f'{sys.path}')
58 from readData import *
59
60 POSITION_RANDOM_PROB = (len(mut_pos)+len(caton_epitopes))/566 if parser.
    POSITION_RANDOM_PROB is None else parser.POSITION_RANDOM_PROB
61 POSITION_RANDOM_PROB = [POSITION_RANDOM_PROB, 1-POSITION_RANDOM_PROB] #
    Likelihood of position being random is first number compared to being from
     dist
62 AA_RANDOM_PROB = (1-np.min(np.max(seq_unit_ratio_df, axis=0))) if parser.
    AA_RANDOM_PROB is None else parser.AA_RANDOM_PROB
63 AA_RANDOM_PROB = [AA_RANDOM_PROB, 1-AA_RANDOM_PROB] # Likelihood of amino acid
     (AA) being random is first number compared to being from dist
64
65 time_taken_ls.append(dt.now()-start_time)
66 print('DONE -\t', time_taken_ls[-1])
67 start_time = dt.now()
68
69 print('Setting up q-learning environment...')
70 # Create directory for qtables
71 data_dir = f'{main_dir}/data/GISAID_1990_2020_H1N1_USA_Human_Protein_HA/'
72 filename = "gisaid_epiflu_sequence_unique_aligned.fasta"
```

```
73 date_time = dt.now().strftime("%Y%m%d-%H%M%S") if parser.datetime is None else
       parser.datetime
74 directory = data_dir+f'qtables_{date_time}'
75 if not os.path.exists(directory):
76     os.makedirs(directory)
77
78 seq_ls = [seq for seq in seq_dict_all.keys()]
79
80 current_season = '2008-2009'
81 next_season = str(int(current_season.split('-')[0])+1) + "-" + str(int(
       current_season.split('-')[1])+1)
82 unit = len(encoder_dict_amino) - 1 # since no aligned there should be no
       insertions or deletions
83 # seq_dict_by_season, seq_dict_all, seq_ls_all are defined in readData.py
84 start_seq = random.choice(seq_dict_by_season[current_season])
85 env = gym.make("MutationSpace-v0",
86                seq_dict_all=seq_dict_all,
87                unit=unit,
88                seq_len=len(start_seq.sequence_aligned.split("\'")[1]),
89                curr_seq=start_seq,
90                is_done_reward=10.0)
91 env.set_consensus_seq(consensus_seq)
92 env.reset(start_seq)
93
94 DISCRETE_OS_SIZE = len(seq_dict_all) # total known sequences
95
96 q_table = np.random.uniform(low=0, high=1, size=(DISCRETE_OS_SIZE,
97                                                  env.action_space.spaces
       [0].n
98                                                   *env.action_space.
       spaces[1].n))
```

```
99
100 time_taken_ls.append(dt.now()-start_time)
101 print('DONE -\t', time_taken_ls[-1])
102
103 print('Starting q-learning...')
104 start_time = dt.now()
105
106 done_dict = {}
107 seq_visited = {key: 0 for key, _ in seq_dict_all.items()}
108
109 """
110 LEARNING
111 """
112 season_ls = ['2008-2009', '2009-2010', '2010-2011', '2011-2012', '2012-2013',
    '2013-2014', '2014-2015', '2015-2016', '2016-2017', '2017-2018','2018-2019
    ']
113 mut_rewards = {}
114 ep_rewards_dict = {}
115 for season in season_ls:
116     done_dict[season] = []
117     current_season = season
118     start_seq = random.choice(seq_dict_by_season[current_season])
119     next_season = str(int(current_season.split('-')[0])+1) + "-" + str(int(
    current_season.split('-')[1])+1)
120     MAX_MUTATIONS = int(len(seq_dict_by_season[next_season])*FRAC_ITERS_MUT)
121     env.set_max_mutations(MAX_MUTATIONS)
122     is_done_ls = seq_dict_by_season[next_season]
123     env.set_is_done_ls(is_done_ls)
124     THRES_DONE = thres_done_dict[next_season] if parser.THRES_DONE is None
    else parser.THRES_DONE # levenshtein distance at which considered done -
    average lv distance per next season (maybe min)
```

```
125     env.set_thres_done(THRES_DONE)
126
127     EXPLORE_PROB = 1 if parser.EXPLORE_PROB is None else parser.EXPLORE_PROB
128     EXPLORE_EXPLOIT_PROB = [EXPLORE_PROB, 1-EXPLORE_PROB] # Explore likelihood
        is first number and Exploit likelihood is second number
129     EPISODES = int(len(seq_dict_by_season[current_season])*FRAC_ITERS_EP)
130     MIN_RANDOM = int(FRAC_MIN_RANDOM*EPISODES)
131     END_EPSILON_DECAYING = EPISODES // 2
132     epsilon_decay_value = EXPLORE_PROB/(END_EPSILON_DECAYING - 1)
133     ep_rewards = []
134     # ep - num episode, avg - average reward, min - worst model, max - best
       model in those episodes
135     aggr_ep_rewards = {'ep': [], 'avg': [], 'min': [], 'max': []}
136
137     output_str = ""
138     counter_best = 0
139     counter_random = 0
140     time_per_mutation_ls = []
141     SHOW_EVERY = EPISODES//10
142     SHOW_EVERY = SHOW_EVERY if SHOW_EVERY != 0 else 1
143     mut_rewards[season] = {}
144     print(current_season)
145     for episode in range(EPISODES):
146         mut_rewards[season][episode] = []
147         dist_amino_acids = seq_unit_ratio_df_dict[current_season]
148         episode_reward = 0
149         if episode % SHOW_EVERY == 0:
150             render = False
151         else:
152             render = False
153
```

```
154         discrete_seq = random.choices(seq_dict_by_season[current_season],
    seq_dict_by_season_prob[current_season])[0]
155         env.reset(discrete_seq)
156         discrete_state = seq_ls.index(discrete_seq.sequence_aligned)
157         env.curr_seq.set_state_idx(discrete_state)
158         done = False
159         num_steps = 0
160 
161         curr_seq_pos = -1
162         while done != 1 and num_steps < env.MAX_MUTATIONS:
163             start_time_mut = dt.now()
164             explore = random.choices([True, False], EXPLORE_EXPLOIT_PROB)[0]
165             aa_rand = random.choices([True, False], AA_RANDOM_PROB)[0]
166             pos_rand = random.choices([True, False], POSITION_RANDOM_PROB)[0]
167             if not explore: # Exploitation
168                 q_values = q_table[discrete_state]
169                 curr_seq_aa = discrete_seq.state[curr_seq_pos]
170                 curr_action_value = (curr_seq_pos*unit) + curr_seq_aa
171                 action_value = np.argmax(q_values)
172                 if action_value == curr_action_value:
173                     action_value = np.argsort(q_values)[1]
174                 action = (action_value%unit, action_value//unit)
175                 pos = action_value//unit
176                 counter_best = counter_best + 1
177             elif aa_rand:
178                 if pos_rand:
179                     unique_pos = np.setdiff1d(range(0, env.action_space.spaces
    [1].n), curr_seq_pos)
180                     pos = random.choices(list(unique_pos))[0]
181                 else:
182                     unique_pos = np.setdiff1d(mut_pos, curr_seq_pos)
```

```
183                     pos = random.choices(list(unique_pos))[0]
184                 curr_seq_aa = discrete_seq.state[pos]
185                 unique_aa = np.setdiff1d(range(0, env.action_space.spaces[0].n
    ), curr_seq_aa)
186                 action = (random.choices(unique_aa)[0], pos)
187                 counter_random = counter_random + 1
188             else:
189                 if pos_rand:
190                     unique_pos = np.setdiff1d(range(0, env.action_space.spaces
    [1].n), curr_seq_pos)
191                     pos = random.choices(list(unique_pos))[0]
192                 else:
193                     unique_pos = np.setdiff1d(mut_pos, curr_seq_pos)
194                     pos = random.choices(list(unique_pos))[0]
195                 curr_seq_aa = discrete_seq.state[pos]
196                 unique_aa = np.setdiff1d(range(0, env.action_space.spaces[0].n
    ), curr_seq_aa)
197                 dist_aa = dist_amino_acids[pos].loc[unique_aa]
198                 dist_aa = dist_aa + 1/len(unique_aa) if sum(dist_aa) == 0.0
    else dist_aa
199                 action = (random.choices(list(unique_aa), dist_aa)[0], pos)
200                 counter_random = counter_random + 1
201
202             curr_seq_pos = pos
203
204             new_seq, reward, done, _ = env.step(action)
205             mut_rewards[season][episode].append(reward)
206
207             num_diff = 0
208             diff_ls = []
209             for k in range(0,len(new_seq.sequence_aligned)):
```

```
210                 if new_seq.sequence_aligned[k] != discrete_seq.
    sequence_aligned[k]:
211                     num_diff = num_diff + 1
212                     diff_ls.append((discrete_seq.sequence_aligned[k], new_seq.
    sequence_aligned[k],k))
213 
214             output_str = output_str + "\n"+ f'Current State: {discrete_seq.
    sequence_aligned}\nNext State: {new_seq.sequence_aligned}\nAction: {action
    }\nReward: {reward}\nDone: {done}'
215             episode_reward += reward
216             num_steps += 1
217 
218             if new_seq.sequence_aligned not in seq_ls:
219                 q_table = np.append(q_table, np.random.uniform(low=0,high=1,
    size=(1,env.action_space.spaces[0].n*env.action_space.spaces[1].n)), axis
    =0)
220                 new_seq.set_state_idx(q_table.shape[0]-1)
221                 seq_dict_all[new_seq.sequence_aligned] = {}
222                 seq_visited[new_seq.sequence_aligned] = 1
223                 seq_dict_all[new_seq.sequence_aligned][new_seq.season] = [
    new_seq]
224                 env.seq_dict_all = seq_dict_all
225                 seq_ls = [seq for seq in seq_dict_all.keys()]
226             else:
227                 new_seq.set_state_idx(seq_ls.index(new_seq.sequence_aligned))
228                 seq_visited[new_seq.sequence_aligned] = seq_visited[new_seq.
    sequence_aligned] + 1
229 
230             new_discrete_state = new_seq.state_idx
231             if done != 1:
232                 max_future_q = np.max(q_table[new_discrete_state]) # max_a Q(
```

```
s_t+1, a)
233                 current_q = q_table[discrete_state, action[0]+(env.
action_space.spaces[0].n*action[1])] # Q(s_t, a_t)
234                 new_q = ((1-LEARNING_RATE)*current_q) + (LEARNING_RATE * (
reward + (DISCOUNT * max_future_q)))
235                 q_table[discrete_state, action[0]+(env.action_space.spaces[0].
n*action[1])] = new_q
236
237             discrete_state = new_discrete_state
238             discrete_seq = new_seq
239             time_per_mutation_ls.append(dt.now()-start_time_mut)
240             start_time_mut = dt.now()
241             done_dict[season].append(done)
242
243         if END_EPSILON_DECAYING >= episode >= MIN_RANDOM:
244             EXPLORE_PROB -= epsilon_decay_value
245             EXPLORE_EXPLOIT_PROB = [EXPLORE_PROB, 1-EXPLORE_PROB]
246
247         ep_rewards.append(episode_reward)
248
249         if not episode % SHOW_EVERY:
250             np.save(open(data_dir+f"qtables_{date_time}/{season}_{episode}
_qtable.npy", "wb"), q_table)
251             average_reward = sum(ep_rewards[-SHOW_EVERY:])/len(ep_rewards[-
SHOW_EVERY:])
252             aggr_ep_rewards['ep'].append(episode)
253             aggr_ep_rewards['avg'].append(average_reward)
254             aggr_ep_rewards['min'].append(min(ep_rewards[-SHOW_EVERY:]))
255             aggr_ep_rewards['max'].append(max(ep_rewards[-SHOW_EVERY:]))
256
257             print(f"Episode: {episode} avg: {average_reward} min: {min(
```

```
    ep_rewards[-SHOW_EVERY:])} max: {max(ep_rewards[-SHOW_EVERY:])}")
258
259     ep_rewards_dict[season] = ep_rewards
260     time_taken_ls.append(dt.now()-start_time)
261     print('DONE -\t', time_taken_ls[-1])
262
263     start_time = dt.now()
264
265     print('Saving inferred results...')
266     with open(data_dir+f'qtables_{date_time}/output_str_{season}.txt', "w") as
        f:
267         f.write(output_str)
268
269     plt.plot(aggr_ep_rewards['ep'], aggr_ep_rewards['avg'], label='avg')
270     plt.plot(aggr_ep_rewards['ep'], aggr_ep_rewards['min'], label='min')
271     plt.plot(aggr_ep_rewards['ep'], aggr_ep_rewards['max'], label='max')
272     plt.legend(loc=4)
273     plt.savefig(data_dir+f'qtables_{date_time}/aggr_ep_rewards_{season}.png')
274
275     output_str_ls = output_str.split('\n')[1:]
276     output_df = pd.DataFrame(columns=['Current State', 'Next State', 'Action -
        AA', 'Action - Pos', 'Reward', 'Done', 'Mutation No.', 'Episode'])
277     curr_state_ls = []
278     next_state_ls = []
279     action_aa_ls = []
280     action_pos_ls = []
281     reward_ls = []
282     done_ls = []
283     mut_num_ls = []
284     ep_ls = []
285
```

```
286     count_i = 0
287     for i in range(0, len(output_str_ls), 5):
288         curr_seq = output_str_ls[i].split(': ')[1]
289         curr_seq = seq_df[seq_df.Sequence == curr_seq].Sequence_Aligned.iloc
    [0] if len(seq_df[seq_df.Sequence == curr_seq].Sequence_Aligned) > 0 else
    curr_seq
290         curr_seq = curr_seq.split('\'')[1]
291         curr_state_ls.append(curr_seq)
292 
293         next_seq = output_str_ls[i+1].split(': ')[1]
294         next_seq = seq_df[seq_df.Sequence == next_seq].Sequence_Aligned.iloc
    [0] if len(seq_df[seq_df.Sequence == next_seq].Sequence_Aligned) > 0 else
    next_seq
295         next_seq = next_seq.split('\'')[1]
296         next_state_ls.append(next_seq)
297 
298         action = output_str_ls[i+2].split(': ')[1].split(', ')
299         action_aa_ls.append(decoder_dict_amino[int(action[0].split('(')[1])])
300         action_pos_ls.append(action[1].split(')')[0])
301 
302         reward_ls.append(output_str_ls[i+3].split(': ')[1])
303         done_ls.append(output_str_ls[i+4].split(': ')[1])
304         mut_num_ls.append(count_i%MAX_MUTATIONS)
305         ep_ls.append(count_i//MAX_MUTATIONS)
306         count_i = count_i + 1
307 
308     output_df['Current State'] = curr_state_ls
309     output_df['Next State'] = next_state_ls
310     output_df['Action - AA'] = action_aa_ls
311     output_df['Action - Pos'] = action_pos_ls
312     output_df['Reward'] = reward_ls
```

```
313     output_df['Done'] = done_ls
314     output_df['Mutation No.'] = mut_num_ls
315     output_df['Episode'] = ep_ls
316     output_df.to_csv(data_dir+f"qtables_{date_time}/output_{season}.csv",
        index=False)
317 
318     pickle.dump(seq_dict_all, open(data_dir+f'qtables_{date_time}/{season}
        _seq_dict_all_after_learning.pkl', 'wb'))
319 
320     time_taken_ls.append(dt.now()-start_time)
321     print('DONE -\t', time_taken_ls[-1])
322 
323 pickle.dump(ep_rewards_dict, open(data_dir+f'qtables_{date_time}/
        ep_rewards_dict.pkl', 'wb'))
324 pickle.dump(mut_rewards, open(data_dir+f'qtables_{date_time}/mut_rewards.pkl',
         'wb'))
325 pickle.dump(seq_visited, open(data_dir+f'qtables_{date_time}/seq_visited.pkl',
         'wb'))
326 
327 # https://pythonprogramming.net/q-learning-analysis-reinforcement-learning-
        python-tutorial/?completed=/q-learning-algorithm-reinforcement-learning-
        python-tutorial/ used as baseline for this q-learning code
328 
329 """
330 PREDICTING
331 """
332 for season in season_ls[:-1]:
333     seq_dict_all = pickle.load(open(data_dir+f'{season}
        _seq_dict_all_after_learning.pkl', 'rb'))
334     all_seq_objs_by_year = [val for key, val in seq_dict_all.items()]
335     temp = [list(val.values())[0] for val in all_seq_objs_by_year]
```

```
336     all_sequence_objects = [item for sublist in temp for item in sublist]
337 
338     all_seq_obj_df = pd.DataFrame()
339     for val in all_sequence_objects:
340         tmp_df = val.to_DataFrame()
341         all_seq_obj_df = pd.concat([all_seq_obj_df, tmp_df])
342     all_seq_obj_df.to_csv(data_dir+f"seq_objects_{season}.csv", index=False)
343 
344     all_seq_obj_df['sequence_aligned_clean'] = all_seq_obj_df['
    sequence_aligned'].apply(lambda x: x.split("\'")[1])
345     all_seq_obj_df_current = all_seq_obj_df.add_suffix(" - Current")
346     output_seq_prop_df = output_df.merge(all_seq_obj_df_current, left_on = "
    Current State", right_on = "sequence_aligned_clean - Current")
347     all_seq_obj_df_next = all_seq_obj_df.add_suffix(" - Next")
348     output_seq_prop_df = output_seq_prop_df.merge(all_seq_obj_df_next, left_on
    = "Next State", right_on = "sequence_aligned_clean - Next")
349     output_seq_prop_df.to_csv(data_dir+f"output_with_properties_{season}.csv",
     index=False)
350 
351 data_dir = f'{main_dir}/data/GISAID_1990_2020_H1N1_USA_Human_Protein_HA/
    qtables_{date_time}/'
352 path_by_season = {}
353 pred_seq_dict = {}
354 for season in season_ls[:-1]:
355     seq_dict_all = pickle.load(open(data_dir+f'{season}
    _seq_dict_all_after_learning.pkl', 'rb'))
356     seq_ls = [seq for seq in seq_dict_all.keys()]
357     path_all = {}
358     pred_seq_ls = []
359     next_season = str(int(season.split('-')[0])+1) + "-" + str(int(season.
    split('-')[1])+1)
```

```
360     filename = f'{next_season}_0_qtable.npy'
361     q_table = np.load(open(data_dir+filename, 'rb'))
362     print(f'Starting {season}')
363     optimal_path_str = ""
364     for seq in seq_dict_by_season[next_season]:
365         path_all[seq] = []
366         curr_state = seq.sequence_aligned
367         path_all[seq].append(curr_state)
368         curr_index = seq_ls.index(curr_state) if curr_state in seq_ls else -1
369         count = 0
370         optimal_path_str = optimal_path_str + 'Start seq:\n' + curr_state + '\
    n'
371         #print('Start seq:\n', curr_state, '\n')
372         while curr_index != -1 and count <= 20:
373             best_action = np.argmax(q_table[curr_index,:])
374             best_action = (best_action%unit, best_action//unit)
375             next_state = curr_state[0:(2+best_action[1])] + decoder_dict_amino
    [best_action[0]] + curr_state[3+best_action[1]:]
376             curr_index = seq_ls.index(next_state) if next_state in seq_ls else
     -1
377             path_all[seq].append(next_state)
378             count = count + 1
379             optimal_path_str += '\t\taction:\t' + decoder_dict_amino[
    best_action[0]] + ' at pos:\t' + str(best_action[1]) + '\n'
380         optimal_path_str += '\t\tPred Seq:\t' + next_state + '\n'
381         pred_seq_ls.append(next_state)
382     with open(data_dir+f'optimal_path_str_{season}.txt', "w") as f:
383         f.write(optimal_path_str)
384     path_by_season[season] = path_all
385     pred_seq_dict[season] = pred_seq_ls
386 pickle.dump(path_by_season, open(data_dir+'path_by_season.pkl', 'wb'))
```

```
387 pickle.dump(pred_seq_dict, open(data_dir+'pred_seq_dict.pkl', 'wb'))
388
389 data_dir = f'{main_dir}/data/GISAID_1990_2020_H1N1_USA_Human_Protein_HA/
    qtables_{date_time}/'
390 lv_dist_df_pred = pd.DataFrame(columns=['Season', 'Pred Seq 1', 'Pred Seq 2',
    'LV Dist'])
391 lv_dist_df = pd.DataFrame(columns=['Season', 'Act Seq', 'Pred Seq', 'LV Dist'
    ])
392 best_pred_for_act = pd.DataFrame(columns=['Season', 'Act Seq', 'Pred Seq', 'LV
     Dist'])
393 lv_dist_df_act = pd.DataFrame(columns=['Season', 'Act Seq 1', 'Act Seq 2', 'LV
     Dist'])
394 lv_dist_df_act = lv_dist_df_act[lv_dist_df_act.Season.isin(season_ls[:-3])]
395 lv_dist_df = lv_dist_df[lv_dist_df.Season.isin(season_ls[:-3])]
396 best_pred_for_act = best_pred_for_act[best_pred_for_act.Season.isin(season_ls
    [:-3])]
397 for season in season_ls[:-1]:
398     pred_season = str(int(season.split('-')[0])+2) + "-" + str(int(season.
    split('-')[1])+2)
399     act_seq_ls = [seq.sequence_aligned for seq in seq_dict_by_season[
    pred_season]]
400     for act_seq in act_seq_ls:
401         for pred_seq in pred_seq_dict[season]:
402             tmp_df = pd.DataFrame({'Season':[season],
403                                    'Act Seq':[act_seq],
404                                    'Pred Seq':[pred_seq],
405                                    'LV Dist':[lv.distance(act_seq, pred_seq)
    ]})
406             lv_dist_df = pd.concat([lv_dist_df,tmp_df])
407         tmp_df = lv_dist_df.loc[(lv_dist_df.Season == season) & (lv_dist_df['
    Act Seq'] == act_seq)]
```

```
408         best_pred_for_act = pd.concat([best_pred_for_act, tmp_df[tmp_df['LV
    Dist'] == tmp_df['LV Dist'].min()]])
409
410     for pred_seq_1 in pred_seq_dict[season]:
411         for pred_seq_2 in pred_seq_dict[season]:
412             if pred_seq_1 != pred_seq_2:
413                 tmp_df = pd.DataFrame({'Season':[season],
414                                        'Pred Seq 1':[pred_seq_1],
415                                        'Pred Seq 2':[pred_seq_2],
416                                        'LV Dist':[lv.distance(pred_seq_1,
    pred_seq_2)]})
417                 lv_dist_df_pred = pd.concat([lv_dist_df_pred,tmp_df])
418
419     for act_seq_1 in act_seq_ls:
420         for act_seq_2 in act_seq_ls:
421             if act_seq_1 != act_seq_2:
422                 tmp_df = pd.DataFrame({'Season':[season],
423                                        'Act Seq 1':[act_seq_1],
424                                    'Act Seq 2':[act_seq_2],
425                                    'LV Dist':[lv.distance(act_seq_1,
    act_seq_2)]})
426                 lv_dist_df_act = pd.concat([lv_dist_df_act,tmp_df])
427
428     plt.hist(lv_dist_df_act[lv_dist_df_act.Season == season]['LV Dist'], alpha
    =0.5, color='blue', label='Actual', density=True)
429     plt.hist(lv_dist_df_pred[lv_dist_df_pred.Season == season]['LV Dist'],
    alpha=0.5, color='orange', label='Predicted', density=True)
430     plt.hist(lv_dist_df[lv_dist_df.Season == season]['LV Dist'], alpha=0.5,
    color='pink', label='Act vs. Pred', density=True)
431     plt.title(f'Amino Acid Distance between Seq - {pred_season}')
432     plt.legend()
```

```
433     plt.savefig(data_dir + f'aa_dist_{pred_season}.png')
434     plt.show()
435 lv_dist_df_pred.reset_index(drop=True, inplace=True)
436 lv_dist_df_pred.to_csv(data_dir+'lv_dist_df_pred.csv', index=False)
437 
438 lv_dist_df.reset_index(drop=True, inplace=True)
439 lv_dist_df.to_csv(data_dir+'lv_dist_df.csv', index=False)
440 best_pred_for_act.reset_index(drop=True, inplace=True)
441 best_pred_for_act.to_csv(data_dir+'best_pred_for_act.csv', index=False)
442 lv_dist_df_act.reset_index(drop=True, inplace=True)
443 lv_dist_df_act.to_csv(data_dir+'lv_dist_df_act.csv', index=False)
444 
445 # KS 2 Sample Test for difference in distribution
446 ks_df = pd.DataFrame(columns=['Season', 'Overall - Statistic', 'Overall - P-
        value', 'Best - Statistic', 'Best - P-value'])
447 overall_dist_statistic = []
448 overall_dist_pvalue = []
449 pred_vs_act_dist_statistic = []
450 pred_vs_act_dist_pvalue = []
451 best_dist_statistic = []
452 best_dist_pvalue = []
453 for season in season_ls[:-3]:#:-1]:
454     tmp_stat, tmp_p = ks_2samp(lv_dist_df_act[lv_dist_df_act.Season == season
        ]['LV Dist'], lv_dist_df[lv_dist_df.Season == season]['LV Dist'])
455     overall_dist_statistic.append(tmp_stat)
456     overall_dist_pvalue.append(tmp_p)
457     tmp_stat, tmp_p = ks_2samp(lv_dist_df_act[lv_dist_df_act.Season == season
        ]['LV Dist'], best_pred_for_act[best_pred_for_act.Season == season]['LV
        Dist'])
458     best_dist_statistic.append(tmp_stat)
459     best_dist_pvalue.append(tmp_p)
```

```
460 ks_df['Season'] = season_ls[:-3]#:-1]
461 ks_df['Overall - Statistic'] = overall_dist_statistic
462 ks_df['Overall - P-value'] = overall_dist_pvalue
463 ks_df['Best - Statistic'] = best_dist_statistic
464 ks_df['Best - P-value'] = best_dist_pvalue
```

Listing C.1: Q-Learning Learning and Prediction

```
1  import gym
2  from gym import Env
3  from gym import spaces
4  import random
5  import numpy as np
6  from gym_mutational_pathways.envs.sequence import Sequence
7  import uuid
8  import Levenshtein as lv
9
10 class CustomEnv(Env):
11     """Custom Environment that follows gym interface"""
12     metadata = {'render.modes': ['human']}
13
14     def __init__(self, seq_dict_all, unit=20, seq_len=600, curr_seq=None,
    is_done_reward=10.0, max_mutations=100):
15         self.action_space = spaces.Tuple((spaces.Discrete(unit),spaces.
    Discrete(seq_len)))
16         self.observation_space = spaces.MultiDiscrete([unit]*seq_len)
17         self.MAX_MUTATIONS = max_mutations
18         self.curr_seq = curr_seq if curr_seq else Sequence([random.sample(
    range(-1, unit-1), 1)[0] for _ in range(seq_len)])
19         self.is_done_reward = is_done_reward
20         self.is_done_ls = []
21         self.encoder_dict_amino = {'A':0,
```

```
22                                      'R':1,
23                                      'N':2,
24                                      'D':3,
25                                      'C':4,
26                                      'Q':5,
27                                      'E':6,
28                                      'G':7,
29                                      'H':8,
30                                      'I':9,
31                                      'L':10,
32                                      'K':11,
33                                      'M':12,
34                                      'F':13,
35                                      'P':14,
36                                      'S':15,
37                                      'T':16,
38                                      'W':17,
39                                      'Y':18,
40                                      'V':19,
41                                      '-':-1}
42          self.decoder_dict_amino = {v: k for k, v in self.encoder_dict_amino.
    items()}
43          self.seq_dict_all = seq_dict_all
44          self.unique_seqs = [seq for seq in self.seq_dict_all.keys()]
45          self.thres_done = 1
46
47      def set_is_done_ls(self, is_done_ls):
48          self.is_done_ls = is_done_ls
49
50      def set_thres_done(self, thres_done):
51          self.thres_done = thres_done
```

```
52
53      def set_consensus_seq(self, consensus_seq):
54          self.consensus_seq = consensus_seq
55
56      def set_max_mutations(self, max_mutations):
57          self.MAX_MUTATIONS = max_mutations
58
59      def step(self, action):
60          new_state = self.curr_seq.state
61          new_state[action[1]] = action[0]
62
63          sequence_aligned = self.curr_seq.sequence_aligned
64          sequence_aligned = sequence_aligned[0:(2+action[1])] + self.
    decoder_dict_amino[action[0]] + sequence_aligned[3+action[1]:]
65
66          if sequence_aligned in self.seq_dict_all.keys():
67              if self.curr_seq.season in self.seq_dict_all[sequence_aligned].
    keys():
68                  next_seq = random.sample(self.seq_dict_all[sequence_aligned][
    self.curr_seq.season],1)[0]
69              elif len(self.seq_dict_all[sequence_aligned]) == 1:
70                  key_season = list(self.seq_dict_all[sequence_aligned].keys())
    [0]
71                  next_seq = random.sample(self.seq_dict_all[sequence_aligned][
    key_season],1)[0]
72              else:
73                  key_seasons = list(self.seq_dict_all[sequence_aligned].keys())
74                  key_seasons.append(self.curr_seq.season)
75                  key_seasons.sort()
76                  curr_season_index = key_seasons.index(self.curr_seq.season)
77                  key_season = key_seasons[curr_season_index+1] if
```

```
    curr_season_index < len(key_seasons) - 1 else key_seasons[
    curr_season_index-1]
78              next_seq = random.sample(self.seq_dict_all[sequence_aligned][
    key_season],1)[0]
79          else:
80              sequence = sequence_aligned.split("\'")[1]
81              for i in range(0, len(sequence)):
82                  if sequence[i] == '-':
83                      sequence = sequence[0:i] + self.decoder_dict_amino[self.
    consensus_seq[i]] + sequence[i+1:]
84              closest_seq = self.similarity(sequence)
85              next_seq = Sequence(state = new_state,
86                                  id = uuid.uuid4(),
87                                  date = closest_seq.date,
88                                  year = closest_seq.year,
89                                  name = '',
90                                  sequence = sequence,
91                                  sequence_aligned = sequence_aligned,
92                                  isolate_id = '',
93                                  lineage = '',
94                                  length = len(new_state),
95                                  strain_freq = 0,
96                                  vaccine_strains = closest_seq.vaccine_strains,
97                                  vaccine_dose = closest_seq.vaccine_dose,
98                                  vaccine_dose_hist = closest_seq.
    vaccine_dose_hist,
99                                  mortality = closest_seq.mortality,
100                                 mortality_v2 = closest_seq.mortality_v2,
101                                 mortality_ratio_v2 = closest_seq.
    mortality_ratio_v2,
102                                 osel_resist_pheno = closest_seq.
```

```
    osel_resist_pheno,
103                                 zana_resist_pheno = closest_seq.
    zana_resist_pheno,
104                                 antigen_character = '',
105                                 season = closest_seq.season,
106                                 hemisphere = closest_seq.hemisphere,
107                                 instability_index = 40,
108                                 total_strains_for_season = closest_seq.
    total_strains_for_season,
109                                 total_population = closest_seq.
    total_population,
110                                 total_population_hist = closest_seq.
    total_population_hist,
111                                 total_infected = closest_seq.total_infected,
112                                 norm_dict = closest_seq.norm_dict,
113                                 num_mut_points = 24,
114                                 vaccine_mut_pos = closest_seq.vaccine_mut_pos,
115                                 all_vaccine_strains = closest_seq.
    all_vaccine_strains,
116                                 mut_pos = closest_seq.mut_pos,
117                                 caton_epitopes = closest_seq.caton_epitopes,
118                                 enc = closest_seq.enc,
119                                 cat_features = closest_seq.cat_features,
120                                 features = closest_seq.features,
121                                 fitness_model = closest_seq.fitness_model)
122             next_seq.set_instability_index()
123
124         reward, done = self.get_reward(self.curr_seq, action, next_seq)
125         self.curr_seq = next_seq
126         info = {}
127         return next_seq, reward, done, info
```

```
128     # add new_state to q_table if not already there
129
130     def similarity(self, seq_new):
131         seq_new_dist = []
132         for seq in self.unique_seqs:
133             seq = seq.split("\'")[1]
134             seq_new_dist.append(lv.distance(seq, seq_new))
135         seq_closest = self.seq_dict_all[self.unique_seqs[np.argsort(np.array(
    seq_new_dist))[0]]]
136         seq_seasons = list(seq_closest.keys())
137         seq_seasons.append(self.curr_seq.season)
138         seq_seasons.sort()
139         curr_season_index = seq_seasons.index(self.curr_seq.season)
140         seq_season = seq_seasons[curr_season_index+1] if curr_season_index <
    len(seq_seasons) - 1 else seq_seasons[curr_season_index-1]
141         return random.sample(seq_closest[seq_season], 1)[0]
142
143     def reset(self, new_curr_seq):
144         self.curr_seq = new_curr_seq
145         return self.curr_seq  # reward, done, info can't be included
146
147     # def render(self, mode='human', close=False):
148     #     self.mutation_space.view()
149
150     # def close (self):
151
152     """
153     Return reward and if done
154     """
155     def get_reward(self, curr_seq, action, next_seq):
156         done = self.is_done(next_seq)
```

```
157         next_seq_fitness = next_seq.get_fitness() if np.isnan(next_seq.fitness
    ) else next_seq.fitness
158         curr_seq_fitness = curr_seq.get_fitness() if np.isnan(curr_seq.fitness
    ) else curr_seq.fitness
159         reward = next_seq_fitness - curr_seq_fitness + (done*self.
    is_done_reward)
160         return reward, done
161
162     def get_interaction(self, curr_seq, next_seq):
163         return random.sample(range(0,100),1)[0]
164
165     def is_done(self, next_seq):
166         # if next_seq's season is the season we are trying to predict
167         # training so we know next_seq
168         is_done_dist = [lv.distance(seq.sequence_aligned, next_seq.
    sequence_aligned) for seq in self.is_done_ls]
169         is_done_dist.sort()
170         return 1 if is_done_dist[0] <= self.thres_done else 1/(is_done_dist[0]
     - self.thres_done + 1)
```

Listing C.2: Open AI Gym Custom Environment

```
1  import numpy as np
2  import pandas as pd
3  import random
4  from datetime import datetime as dt
5  from scipy.spatial import distance
6  import Levenshtein as lv
7  from Bio.SeqUtils.ProtParam import ProteinAnalysis
8
9  class Sequence:
10     # for mutated sequences param sequence and param sequence aligned should
```

```
    be the same
    # length - unaligned length
    # if instability_index > 40 then unstable
    def __init__(self, state, id, date=dt(1000,1,1), year=1000, name='',
sequence='', sequence_aligned='',
                 isolate_id='', lineage='', length=566, strain_freq=0,
vaccine_strains=[],
                 vaccine_dose = 0, vaccine_dose_hist=[], mortality=0,
mortality_v2=0,
                 mortality_ratio_v2 = 0, osel_resist_pheno=0,
zana_resist_pheno=0,
                 antigen_character = '', season = '', hemisphere = '',
instability_index=40,
                 total_strains_for_season = 1, total_population = 1,
total_population_hist = [],
                 total_infected = 1, norm_dict = {'':{}}, num_mut_points = 24,
 vaccine_mut_pos = {},
                 all_vaccine_strains = {}, mut_pos = [], caton_epitopes = [],
                 enc = None, cat_features = [], features = [], fitness_model =
 None, fitness_true = None):
        # state is the numerical representation of the aligned sequence
        self.state = state
        self.state_idx = -1 # potentially element later
        self.id = id
        self.date = date
        self.year = year
        self.name = name
        self.isolate_id = isolate_id
        self.lineage = lineage
        self.sequence = sequence
        self.length = length
```

```
33          self.sequence_aligned = sequence_aligned
34          self.strain_freq = strain_freq
35          self.vaccine_strains = vaccine_strains
36          self.vaccine_dose = vaccine_dose
37          self.vaccine_dose_hist = np.array(vaccine_dose_hist)
38          self.mortality = mortality # expressed in percent - by season
39          self.mortality_v2 = mortality_v2
40          self.mortality_ratio_v2 = mortality_ratio_v2
41          self.mortality_ratio_v2_norm = np.nan
42
43          self.osel_resist_pheno = osel_resist_pheno
44          self.zana_resist_pheno = zana_resist_pheno
45          self.antigen_character = antigen_character
46          self.season = season
47          self.hemisphere = hemisphere
48          self.instability_index = instability_index
49          self.total_strains_for_season = total_strains_for_season
50          self.total_population = total_population
51          self.total_population_hist = np.array(total_population_hist)
52          self.total_infected = total_infected
53
54          self.weight_stability = 25
55          self.weight_vaccine = 45
56          self.weight_mortality = 15
57          self.weight_infected = 10
58          self.weight_drugs = 5
59          self.weight_vaccine_strains = np.array([1/i for i in range(1, len(
    vaccine_strains)+1)]) # exponential decay
60          self.weight_freq_est = 0
61
62          self.fitness_mortality = np.nan
```

```
63          self.fitness_infected = np.nan
64          self.fitness_vaccine = np.nan
65          self.fitness_stability = np.nan
66          self.fitness_drugs = np.nan
67          self.fitness = np.nan
68
69          self.norm_dict = norm_dict
70          self.update_norm_dict = norm_dict
71          self.update_norm = False
72          self.num_update_norm = 0
73
74          self.fitness_norm_stability = np.nan
75          self.fitness_norm_vaccine = np.nan
76          self.fitness_norm_mortality = np.nan
77          self.fitness_norm_drugs = np.nan
78
79          self.vaccine_dist = np.nan
80          self.vaccine_dose_effect = np.nan
81          self.num_mut_points = num_mut_points
82          self.vaccine_mut_pos = vaccine_mut_pos
83          self.vaccine_mut_pos_norm = {key:np.nan for key, _ in vaccine_mut_pos.
    items()}
84          self.all_vaccine_strains = all_vaccine_strains
85          self.vaccine_names = [key for key in all_vaccine_strains.keys()]
86
87          self.mut_pos = mut_pos
88          self.mut_pos_seq_ls = []
89          self.caton_epitopes = caton_epitopes
90          self.caton_seq_ls = []
91
92          self.fitness_components_bool = False
```

```
93          self.fitness_model = fitness_model
94          self.enc = enc
95          self.cat_features = cat_features
96          self.features = features
97
98          self.fitness_true = fitness_true
99          self.fitness_true_norm = np.nan
100
101     def set_state_idx(self, state_idx):
102         self.state_idx = state_idx
103
104     def set_vaccine_strains(self, vaccine_strains):
105         self.vaccine_strains = vaccine_strains
106
107     def set_instability_index(self):
108         protein = ProteinAnalysis(self.sequence)
109         self.instability_index = protein.instability_index()
110
111     def set_norm_dict(self, norm_dict):
112         self.norm_dict = norm_dict
113         self.update_norm_dict = norm_dict
114
115     """
116     Note: Alignment cannot be performed every time new seq created so original
        seq dist to
117     vaccine strains is not going to same as new seq dist to vaccine strains
118     """
119     def set_vaccine_dist(self):
120         for name in self.vaccine_names:
121             self.vaccine_mut_pos[name] = self.seq_distance(self.
    all_vaccine_strains[name])
```

```
122
123     # Currently unnecessary but maybe useful
124     def set_pos_seq(self):
125         self.mut_pos_seq_ls = [self.sequence_aligned[pos + 2] for pos in self.
    mut_pos]
126         self.caton_seq_ls = [self.sequence_aligned[pos + 2] for pos in self.
    caton_epitopes]
127
128     def set_fitness_model(self, enc, cat_features, features, fitness_model):
129         self.fitness_model = fitness_model
130         self.enc = enc
131         self.cat_features = cat_features
132         self.features = features
133
134     def set_fitness_true(self, fitness_true):
135         self.fitness_true = fitness_true
136
137     def seq_distance(self, seq2, dis_type="Levenshtein"):
138         seq2_aligned = seq2
139         if isinstance(seq2, Sequence):
140             seq2_aligned = seq2.sequence_aligned
141         if dis_type == "Hamming":
142             dist = distance.hamming(list(self.sequence_aligned),list(
    seq2_aligned))
143             return dist
144         else:
145             dist = lv.distance(self.sequence_aligned,seq2_aligned)
146             return dist
147
148     def seq_diff_positions(self, seq2):
149         seq_diff_ls = []
```

```
150         seq2_aligned = seq2
151         if isinstance(seq2, Sequence):
152             seq2_aligned = seq2.sequence_aligned
153         for i in range(0, len(self.sequence_aligned)):
154             if self.sequence_aligned[i] != seq2_aligned[i]:
155                 seq_diff_ls.append(i)
156         return seq_diff_ls
157 
158     def protein_stability(self):
159         pass
160 
161     def sigmoid(self, x):
162         return 1/(1 + np.exp(-x))
163 
164     def norm(self, x, var_name):
165         x_max = self.norm_dict[var_name+'_max'][self.season]
166         x_min = self.norm_dict[var_name+'_min'][self.season]
167         if x < x_min:
168             self.update_norm = True
169             self.update_norm_dict[var_name+'_min'][self.season] = x
170             self.num_update_norm = self.num_update_norm + 1
171             x = x_min
172         elif x > x_max:
173             self.update_norm = True
174             self.update_norm_dict[var_name+'_max'][self.season] = x
175             self.num_update_norm = self.num_update_norm + 1
176             x = x_max
177         return (x - x_min)/(x_max - x_min)
178 
179     def get_fitness_components(self):
180         # if instability > 40 then unstable which is bad so negative else
```

```
    positive; closer to 0 higher reward
181         self.fitness_mortality = (self.strain_freq/self.
    total_strains_for_season)*self.mortality
182         self.fitness_infected = (self.strain_freq/self.
    total_strains_for_season)*self.total_infected
183
184         self.vaccine_dist = np.array([self.seq_distance(strain) for strain in
    self.vaccine_strains])*self.weight_vaccine_strains
185
186         self.vaccine_dose_effect = -np.log10(self.vaccine_dose_hist/self.
    total_population_hist)
187         self.vaccine_dose_effect = [self.vaccine_dose_effect[i] if not np.
    isnan(self.vaccine_dose) and self.vaccine_dist[i] < self.num_mut_points
    else 1 for i in range(0,len(self.vaccine_dose_effect))]
188         self.fitness_vaccine = np.mean(self.vaccine_dose_effect*self.
    vaccine_dist)
189
190         self.fitness_stability = (40.0 - self.instability_index)
191
192         self.fitness_drugs = (self.osel_resist_pheno)*10 + (self.
    zana_resist_pheno)*10
193         self.fitness_drugs =  self.fitness_drugs if not np.isnan(self.
    osel_resist_pheno) and not np.isnan(self.zana_resist_pheno) else 0
194         self.fitness_components_bool = True
195
196     def get_fitness(self):
197         if self.fitness_true:
198             self.fitness_true_norm = self.norm(self.fitness_true, 'true')
199             self.fitness = self.fitness_true_norm*100
200         elif self.fitness_model:
201             self.fitness_stability = (40.0 - self.instability_index)
```

```
202             # Normalize
203             self.fitness_norm_stability = self.norm(self.fitness_stability, '
    stability')
204             self.mortality_ratio_v2_norm = self.norm(self.mortality_ratio_v2,
    'mortality_ratio_v2')
205             for name in self.vaccine_names:
206                 self.vaccine_mut_pos_norm[name] = self.norm(self.
    vaccine_mut_pos[name], name)
207
208             # Encode Categorical Features
209             X = self.to_DataFrame()
210             X_cat_encoded = self.enc.transform(X[self.cat_features]).toarray()
211             cat_feature_names = self.enc.get_feature_names(self.cat_features)
212             X = pd.concat([X[self.features],
213                            pd.DataFrame(X_cat_encoded,columns=
    cat_feature_names).astype(int)], axis=1)
214
215             # Preict fitness
216             self.fitness = self.fitness_model.predict(X)*100
217         else:
218             # Calculate components
219             if not self.fitness_components_bool:
220                 self.get_fitness_components()
221
222             # Normalize
223             self.fitness_norm_stability = self.norm(self.fitness_stability, '
    stability')
224             self.fitness_norm_vaccine = self.norm(self.fitness_vaccine, '
    vaccine')
225             self.fitness_norm_mortality = self.norm(self.fitness_mortality, '
    mortality')
```

```
226             self.fitness_norm_drugs = self.norm(self.fitness_drugs, 'drugs')
227
228             # Calculate fitness
229             self.fitness = (self.weight_stability*self.fitness_norm_stability)
    \
230                         + (self.weight_vaccine*self.fitness_norm_vaccine) \
231                         + (self.weight_mortality*self.fitness_norm_mortality)
    \
232                         + (self.weight_drugs*self.fitness_norm_drugs)
233         return self.fitness
234
235     def to_DataFrame(self):
236         tmp_df = pd.DataFrame({'state': [str(self.state)],
237                                'state_idx': [str(self.state_idx)],
238                                'id': [str(self.id)],
239                                'date': [str(self.date)],
240                                'year': [str(self.year)],
241                                'name': [str(self.name)],
242                                'isolate_id': [str(self.isolate_id)],
243                                'lineage': [str(self.lineage)],
244                                'sequence': [str(self.sequence)],
245                                'length': [self.length],
246                                'sequence_aligned': [str(self.sequence_aligned)
    ],
247                                'strain_freq': [self.strain_freq],
248                                'vaccine_strains': [str(self.vaccine_strains)],
249                                'vaccine_dose': [self.vaccine_dose],
250                                'vaccine_dose_hist': [self.vaccine_dose_hist],
251                                'mortality': [self.mortality],
252                                'mortality_v2': [self.mortality_v2],
253                                'mortality_ratio_v2': [self.mortality_ratio_v2
```

```
    ],
                                'mortality_ratio_v2_norm': [self.
    mortality_ratio_v2_norm],
                                'osel_resist_pheno': [self.osel_resist_pheno],
                                'zana_resist_pheno': [self.zana_resist_pheno],
                                'antigen_character': [str(self.
    antigen_character)],
                                'season': [str(self.season)],
                                'hemisphere': [str(self.hemisphere)],
                                'instability_index': [self.instability_index],
                                'total_strains_for_season': [self.
    total_strains_for_season],
                                'total_population': [self.total_population],
                                'total_population_hist': [self.
    total_population_hist],
                                'total_infected': [self.total_infected],
                                'weight_stability': [self.weight_stability],
                                'weight_vaccine': [self.weight_vaccine],
                                'weight_mortality': [self.weight_mortality],
                                'weight_infected': [self.weight_infected],
                                'weight_drugs': [self.weight_drugs],
                                'weight_vaccine_strains': [self.
    weight_vaccine_strains],
                                'fitness_mortality': [self.fitness_mortality],
                                'fitness_infected': [self.fitness_infected],
                                'fitness_vaccine': [self.fitness_vaccine],
                                'fitness_stability': [self.fitness_stability],
                                'fitness_drugs': [self.fitness_drugs],
                                'fitness': [self.fitness],
                                'fitness_norm_mortality': [self.
    fitness_norm_mortality],
```

```
278                                 'fitness_norm_vaccine': [self.
    fitness_norm_vaccine],
279                                 'fitness_norm_stability': [self.
    fitness_norm_stability],
280                                 'fitness_norm_drugs': [self.fitness_norm_drugs
    ],
281                                 'vaccine_dist': [self.vaccine_dist],
282                                 'vaccine_dose_effect': [self.
    vaccine_dose_effect],
283                                 'fitness_true': self.fitness_true,
284                                 'fitness_true_norm': self.fitness_true_norm})
285         for name in self.vaccine_names:
286             tmp_df[name] = self.vaccine_mut_pos[name]
287             tmp_df[name+'_norm'] = self.vaccine_mut_pos_norm[name]
288 
289         for pos in self.mut_pos:
290             tmp_df[f'mut_pos_seq_{pos}'] =self.sequence_aligned[pos + 2]
291 
292         for pos in self.caton_epitopes:
293             tmp_df[f'caton_seq_{pos}'] = self.sequence_aligned[pos + 2]
294 
295         return tmp_df
296 
297     def to_dict(self):
298         seq_dict_rep = {}
299         for key, value in self.__dict__.items():
300             seq_dict_rep[key] = value
301         seq_dict_rep['date'] = dt.strftime(seq_dict_rep['date'], '%Y-%m-%d')
    if not isinstance(seq_dict_rep['date'], str) else seq_dict_rep['date']
302         seq_dict_rep['vaccine_dose_hist'] = list(seq_dict_rep['
    vaccine_dose_hist']) if not isinstance(seq_dict_rep['vaccine_dose_hist'],
```

```
    list) else seq_dict_rep['vaccine_dose_hist']
303         seq_dict_rep['total_population_hist'] = list(seq_dict_rep['
    total_population_hist']) if not isinstance(seq_dict_rep['
    total_population_hist'], list) else seq_dict_rep['total_population_hist']
304         seq_dict_rep['weight_vaccine_strains'] = list(seq_dict_rep['
    weight_vaccine_strains']) if not isinstance(seq_dict_rep['
    weight_vaccine_strains'], list) else seq_dict_rep['weight_vaccine_strains'
    ]
305         seq_dict_rep['vaccine_dist'] = list(seq_dict_rep['vaccine_dist']) if
    not isinstance(seq_dict_rep['vaccine_dist'], list) else seq_dict_rep['
    vaccine_dist']
306         seq_dict_rep['fitness_model'] = None
307         seq_dict_rep['enc'] = None
308         return seq_dict_rep
309
310     def to_string(self):
311         return f'State:\t {self.state}\n' + f'state_idx:\t {self.state_idx}\n'
     + \
312                f'id:\t {self.id}\n' + f'date:\t {self.date}\n' + f'year:\t {self.
    year}\n' + \
313                f'name:\t {self.name}\n' + f'isolate_id:\t {self.isolate_id}\n' +
    \
314                f'lineage:\t {self.lineage}\n' + f'sequence:\t {self.sequence}\n'
    + \
315                f'length:\t {self.length}\n' + f'sequence_aligned:\t {self.
    sequence_aligned}\n' + \
316                f'strain_freq:\t {self.strain_freq}\n' + f'vaccine_strains:\t {
    self.vaccine_strains}\n' + \
317                f'vaccine_dose:\t {self.vaccine_dose}\n' + f'mortality:\t {self.
    mortality}\n' + \
318                f'osel_resist_pheno:\t {self.osel_resist_pheno}\n' + \
```

```
319             f'zana_resist_pheno:\t {self.zana_resist_pheno}\n' + \
320             f'antigen_character:\t {self.antigen_character}\n' + f'season:\t {
    self.season}\n' + \
321             f'hemisphere:\t {self.hemisphere}\n' + f'instability_index:\t {
    self.instability_index}\n' + \
322             f'total_strains_for_season:\t {self.total_strains_for_season}\n' +
     \
323             f'weight_stability:\t {self.weight_stability}\n' + \
324             f'weight_vaccine:\t {self.weight_vaccine}\n' + \
325             f'weight_mortality:\t {self.weight_mortality}\n' + f'weight_drugs
    :\t {self.weight_drugs}\n'
```

Listing C.3: Sequence Object Specification

```
1 from gym_mutational_pathways.envs.environment_v2 import CustomEnv
```

Listing C.4: Initialization for Open AI Gym Custom Environment

```
1 from gym.envs.registration import register
2
3 register(
4     id='MutationSpace-v0',
5     entry_point='gym_mutational_pathways.envs:CustomEnv',
6     kwargs={'seq_dict_all':{},'unit':20, 'seq_len':600, 'curr_seq':None, '
    is_done_reward':10.0,'max_mutations':100}
7     )
```

Listing C.5: Initialization for Open AI Gym

```
1 from setuptools import setup
2
3 setup(name='gym_mutational_pathways',
4       version='0.0.1',
```

```
5       install_requires=['gym']  # And any other dependencies foo needs
6 )
```

Listing C.6: Setup for Open AI Gym Custom Environment