Detection of Errors in Multiple Genome Alignments using Machine Learning Approaches

Jaspal Singh

Masters of Science

School of Computer Science
McGill University
Montreal, Quebec, Canada

June 2018

A thesis submitted to McGill University in partial
fulfillment of the requirements of the degree of
Master of Science

Abstract

Multiple sequence alignment is a prerequisite for most evolutionary and phylogenetic analyses. Previous work has shown that existing alignment algorithms are not perfectly accurate, and therefore it is important to identify suspicious regions in whole-genome alignments. In this thesis, we develop a machine learning model that can detect errors in an alignment. The model is trained on data that is obtained by comparing a true alignment file (generated using an alignment simulation tool) with a predicted alignment file (generated using an alignment algorithm). Using the developed method, the sequences from each pair of species at each alignment column can be categorized into one of five possible classes. The training data that consists of the nucleotides at each column, was enhanced by adding relevant features, such as the inferred ancestral nucleotides or features of neighboring alignment columns. Two types of machine learning classifiers, random forests and Artificial Neural Networks (ANN) were trained on the data. ANN performed better and the best accuracy obtained was 65.04%. We applied the model on 1 Mb of real alignment data consisting of ten species from the 100-way alignment available on the UCSC genome browser. Our model predicts 40% of human-cow alignment to be suspiciously aligned. As evidence, we show that the model predicts highly conserved regions to be correctly aligned, while regions with high number of mismatches or regions corresponding to annotated transposable elements are predicted to be suspiciously aligned. Our tool will be useful for researchers who use these alignments for downstream analyses so that they can be cautious of their results on suspiciously aligned regions or find ways to correct these alignments.

Abrégé

L'alignement multiple des séquences est une condition préalable à la plupart des analyses évolutives et phylogénétiques. Des travaux antérieurs ont montré que les algorithmes d'alignement existants ne sont pas parfaitement précis et il est donc important d'identifier les régions suspectes dans les alignements du génome entier. Dans cette thése, nous développons un modèle d'apprentissage automatique capable de détecter les erreurs dans un alignement. Le modèle est entrainé sur les données obtenues en comparant un alignement correct (généré à l'aide d'un outil de simulation d'alignement) avec un alignement prédit (généré à l'aide d'un algorithme d'alignement). En utilisant la méthode développée, les séquences de chaque paire d'espèces à chaque colonne d'alignement peuvent être classées dans l'une de cinq classes possibles. Les données d'entraînement qui se composent des nucléotides à chaque colonne ont été améliorées en ajoutant des caractéristiques pertinentes, telles que les nucléotides ancestraux déduits ou les caractéristiques des colonnes d'alignement voisines. Deux types de classificateurs d'apprentissage automatique, une forêt aléatoire et des réseaux de neurone ont été entrainés sur les données. Les réseaux de neurones ont mieux performé et la meilleure précision obtenue a été de 65.04%. Nous avons appliqué le modèle sur 1 Mb de données d'alignement réel consistant en dix espèces extraites d'un alignment de génomes complets disponible sur le navigateur du génome de l'UCSC. Notre modèle prédit que 40% des colonnes d'alignments entre les séquences humaines et de la vache sont alignés de façon suspecte. Comme preuve, nous montrons que le modèle prédit que les régions hautement conservées sont correctement alignées, tandis que les régions ayant un nombre élevé de mésappariements ou des régions correspondant à des éléments transposables annotés sont prédites être alignés de façon suspecte. Notre outil sera utile pour les chercheurs qui utilisent ces alignements pour des analyses en aval, afin qu'ils puissent se méfier de leurs résultats sur des régions identifiées comme étant suspectes ou trouver des moyens de corriger ces alignements.

Acknowledgements

This thesis would have been impossible to complete without the valuable guidance of my supervisor, Professor Mathieu Blanchette. He introduced me to the field of computational biology and has guided me in every step towards the completion of my master's degree. A special mention to Ramchalam K.R., another student in the Blanchette lab, who has been a great friend and helped me with this project.

I would also like to thank all members of the bioinformatics lab, especially Faizy Ahsan, Mansha Imtiyaz, Chrisopher J.F. Cameron, Ayrin A. Tabibi and Alexander Butyaev for creating a learning and helpful atmosphere in the lab.

Finally, I would like to thank my parents and my girlfriend for their emotional and spiritual support throughout the program.

Contents

List of Figures

List of Tables

1

Introduction

The sequence alignment problem is one of the most fundamental problems in computational biology. It is the starting point of many biological analyses. Sequence alignment algorithms have been developed since the early 1970s. However, it is computationally difficult to optimally align multiple genome length sequences and most of the algorithms used in practice are heuristic-based approximate methods. The use of these imperfect heuristic approaches coupled with some other reasons (discussed later) result in alignment errors. In this thesis, we build a machine learning classifier that can detect errors in the alignment columns of a multiple sequence alignment.

1.1 Background

All living organisms are genetically encoded with deoxyribonucleic acid (DNA) sequences. DNA is composed of four bases (or nucleotides): adenine (A), cytosine (C), guanine (G) and thymine (T). In humans, for example, the DNA is organized as chromosomes. There are 46 chromosomes in the human genome which pair up to form 23 pairs of chromosomes. Out of these 23 pairs, one pair of chromosomes is the sex chromosome and the other 22 pairs are called *autosomes*. In each pair one chromosome comes from the father while the other from the mother, but these chromosomes are not just copied from our parents. There are genetic events such as mutations or cross over that can result in changes of a single nucleotide or multiple nucleotides in the offsprings. Therefore, every DNA (or protein) sequence has an evolutionary history.

1.1 Background

During evolution, new and distinct species are formed by the splitting of one evolutionary lineage into two or more genetically independent lineages. This process is called *speciation*. Combined over million of years, the evolutionary history among many biological species can be represented by a tree-like structure, called phylogenetic tree. A rooted phylogenetic tree as the name suggests is a tree with a root that is also referred to as the most recent common ancestor of the leaves of the tree. Each internal node of the tree represents a speciation event. The edge lengths represent the time of evolution between the species. Figure 1.1 shows a rooted phylogenetic tree. Unrooted phylogenetic trees are used to show the relatedness of the leaves of the trees without any assumption about the ancestry of the leaves.

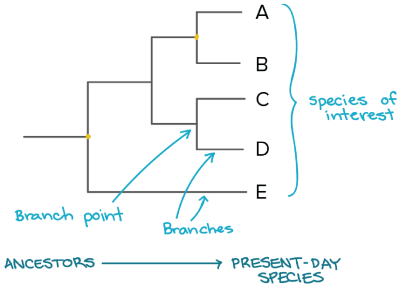


Figure 1.1: A rooted phylogenetic tree. An example of a rooted phylogenetic tree with five leaf species. The important elements are labeled. Figure is taken from [1].

With advancement in sequencing, we now know the entire genome sequence of many species like human [2], mouse [3], etc. One of the best ways to annotate functional elements of the human genome is by comparing it with genomes of other species [4]. Knowing the sequence similarities among sequences of multiple related species can assist in identifying specific sub-sequences that are responsible for various biological functions such as transcription factor binding sites, protein coding regions etc. In other words, *comparative genomics* can assist in annotating genomes, especially the human genome but is dependent on the availability of accurate multiple genome alignments, which are objects we discuss in the next section.

1.2 Multiple Sequence Alignment

The alignment of two or more biological sequences is an arrangement of the sequences in a way that ensures homologous residues or nucleotides are grouped in the same column. To account for the insertions or deletions (together referred as indels) in the sequences we add an additional gap character. A pairwise sequence alignment of a sequence S of length m ($S = S_1, S_2, .., S_m$) with a sequence T of length n ($T = T_1, T_2, .., T_n$) is a pair of strings S' and T' where S' is obtained by inserting gaps in S and T' is obtained by inserting gaps in T such that the length of S' is equal to the length of T'. To compare two alignments we need a way to score them. Pairwise alignments are scored by calculating a match/mismatch score for every alignment column. Matches are rewarded while mismatches are penalized. The scores for mismatches among different nucleotides (substitutions) is represented by a symmetric substitution matrix. The indels are scored by a gap scoring scheme. The most popular gap scoring scheme is the affine gap penalty scheme where the penalty for a gap of size k is $o + (k \times e)$, where o is the gap opening penalty and e is the gap extension penalty. An example of a pairwise alignment of two sequences S1 and S2, along with the substitution matrix used to score the alignment is shown in Figure 1.2.

S1: A C A T
S2: A T - T

Substitution Matrix

	A	C	G	T
A	1	-2	-1	-2
C	-2	1	-2	-1
G	-1	-2	1	-2
T	-2	-1	-2	1

Figure 1.2: Pairwise alignment: An example. The substitution matrix used to score the alignment is shown along side. Considering gap opening and gap extension penalty as -2 and -1 respectively, the score of the alignment is -2 under affine gap penalty scheme.

Pairwise alignments can be generalized to Multiple Sequence Alignments (MSA) when the number of sequences involved is greater than two. An example of a MSA is shown in Figure 1.3.

1.2 Multiple Sequence Alignment

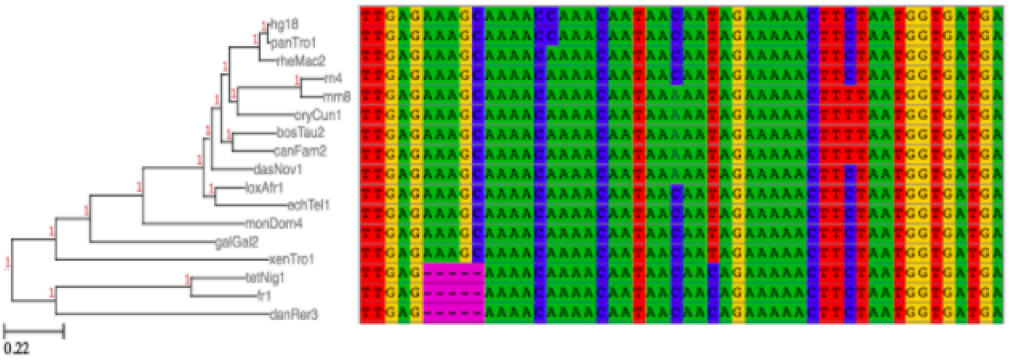


Figure 1.3: An example of a MSA. Figure is taken from [5].

A MSA problem is characterized by the number of sequences, the length of each sequence, the characters that constitute the sequences (DNAs, RNAs or amino acids) and the relatedness of these sequences (phylogenetic tree). For evolutionary correct alignments, the relatedness of these sequences can be defined by finding sequences with a common ancestry (homologous sequences). Formally, if nucleotide i of species X is aligned to nucleotide j of species Y, this implies that i and j are inherited from the same common ancestor. In this thesis, we focus on methods and difficulties related to aligning large genome length (> 1 Mbp) sequences. Obtaining an accurate MSA becomes more difficult if duplication events are considered because multiple nucleotides of the same genome may be homologous to other species. Furthermore, if the species have diverged a lot, differentiating homologous sequences from non homologous sequences become more difficult.

For MSAs, the most popular alignment scoring scheme is the sum-of-pairs alignment score, which is the sum of scores of all $\binom{n}{2}$ pairwise alignments induced by the MSA. Some other variants of the sum-of-pairs-score such as the weighted sum-of-pairs-score, tree alignment score and consensus score [6] are less common in practice. It is shown that MSA is a NP-hard problem [7] when the objective is to maximize the alignment score for most scoring schemes.

Alignment approaches are termed as global when the alignment has to span the entire length of all the sequences. This approach is appropriate when the sequences are closely related to each other and the difference in these sequences would only constitute of indels

or substitutions. Since the entire sequence has to be in one alignment, global approaches cannot accommodate for genomic rearrangements and duplications. Local alignment approaches find homologous alignments among smaller sub-strings of the input sequence and leave rest of the sequence unaligned. This makes the alignments robust enough to handle genomic rearrangement and duplications. However, it increases the risk of getting random hits among smaller alignment blocks and falsely marking them as homologous. Many alignment algorithms take a hybrid approach where they first find local alignments and then combine them as chains of many local alignment. This approach is appropriately termed as *"glocal"* alignments [8]. There are many existing approaches to multiple sequence alignment and these are explained in the next section.

1.3 Review of Existing Whole-genome MSA Algorithms

MSA algorithms can broadly be categorized into two categories: those that guarantee to find the optimal alignments and those that are based on certain heuristics. Before reviewing some multiple sequence alignment algorithms, let us first discuss about pairwise alignments because at the base of most multiple sequence aligners lies a fast pairwise sequence alignment algorithm [4]. Pairwise sequences of moderate size ($\sim$ 100kb) can be optimally aligned by dynamic programming approaches: Needleman-Wunsch [9] for global alignments or Smith-Waterman algorithm [10] for local alignments. The running time of these algorithms is $\mathcal{O}(n^2)$ where n is the average length of all the input sequences. These approaches are not computationally feasible for larger sequences and we need a faster pairwise alignment algorithm. For larger pairwise alignments a heuristic approach called seeded alignments is used. In this heuristic, the algorithms search for local alignment only if they contain some highly conserved short sequence matches. After finding this short match of sequence the alignment is extended using the Smith-Waterman algorithm that allows for indels. BLAST [11], LAGAN [12], AVID [13], MUMmer [14] are examples of seed-based alignments. Multiple local alignment regions are then chained using algorithms such as Chaining/Netting [15], GRIMM-synteny [16], MAUVE [17] and MERCATOR [18]. In this phase, multiple alignment regions can mingle together that allows to find homology between sequences that undergo genomic rearrangements and duplications.

For multiple sequence aligners, some algorithms were developed that determine the

optimal alignment. These were based on extension of dynamic programming techniques ([19, 20, 21, 22]) or on graph formulation algorithms that use combinatorial techniques ([23, 24, 25]). Similar to pairwise optimal aligners, these techniques are computationally infeasible for larger sequences. The running time of such algorithms grows exponentially with respect to the number or length of the sequences to be aligned. Therefore to align larger sequences or a large number of short sequences, heuristic-based approximate algorithms were developed.

The most common form of heuristic for MSA is called progressive sequence multiple alignment [26]. In this heuristic, a phylogenetic tree is inferred (using UPGMA[27] or neighbor-joining method [28]) from the sequences or is provided as input. Based on the tree the two closest species are aligned first, then a third sequence is chosen and aligned to the first alignment without making any changes to the first alignment. This process is repeated till we obtain an alignment with all the species. Some of the famous algorithms that use this heuristic are CLUSTAL [29, 30], TBA [31]/MULTIZ [32, 33], MLAGAN [12] and MAVID [34]. These algorithms differ in their choice of algorithm used for pairwise sequence alignment and their scoring function when they combine two pairs of aligned sequences. TBA/MULTIZ joins two pairwise alignment blocks by combining two pairwise alignments of extant species. MULTIZ was used to produce the genome alignments available in the UCSC Genome Browser [35] including a whole-genome alignment of 100 vertebrate genomes. All these alignments use the seeded pairwise alignment approach for the pairwise alignments. In this study, we use the MULTIZ algorithm which is described in the next section.

1.3.1 MULTIZ

A general approach in aligning multiple genomes is to pick one of the genomes as a "reference genome". All the other sequences are aligned pairwise to this reference genome. This is done primarily to annotate the reference sequence by finding the orthology between all the other sequence and the reference sequence. A drawback of this method is that the conserved regions in the subset of species apart from the reference are not identified. The local pairwise alignments are calculated by BLASTZ and the alignments between three or more sequences is calculated by MULTIZ. MULTIZ combines two or more blocks of

alignments. Consider there are two blocks to combine: Block M with reference genome S and block N with reference genome T. We also have another block G which is the alignment of S and T having the reference S. The result of MULTIZ is an S-ref block for the union of the original sequences in M and N. Figure 1.4 shows this pictorially.

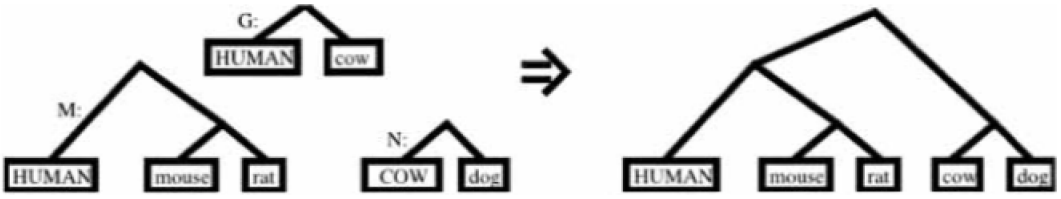


Figure 1.4: Pictorial representation of MULTIZ. M is a human-ref blockset of human, mouse, and rat, whereas N is a cow-ref blockset of cow and dog. MULTIZ uses a pairwise human-ref blockset, G, of human and cow to guide the aligning process. The output is a human-ref blockset of human, mouse, rat, cow, and dog. The reference sequence for each blockset is indicated by capital letters. Figure is taken from [31].

MULTIZ finds segments of a block in M and N that align to each other according to some block g in G. These segments are aligned using dynamic programming.

1.4 Inaccuracies in MSA

In the process of aligning large sequences in reasonable time, most algorithms produce suboptimal alignments. For example, in progressive MSA approach, a pairwise alignment block is not revised with respect to the other species in the alignment, which may result in suboptimal alignments. Researchers have proposed a post processing step to refine the alignments produced. This process is called realignment [36]. One of the realignment strategies used in CLUSTALW and MLAGAN is to remove individual sequences from the alignment and realign them to the remaining $n-1$ sequences. For highly related sequences, a similar approach could be to take the alignment of a subtree and re-align it to all the other sequences. Some other approaches try making local changes to the alignments by moving gaps to optimize an objective function (such as genetic algorithms [37]).

Another post-processing step called *alignment cleaning* is applied where some align-

ment blocks that do not meet a certain statistical thresholds [38] are either removed from the complete alignment or broken down into smaller supported sub-alignments. Most of the alignment algorithms do not give a confidence value with each alignment due to which it is difficult for researches to fully trust these alignments for further biological analyses. This called for a benchmarking standard to be developed for the MSA algorithms. The next section describes some of the benchmarking methods available in the literature.

1.5 Benchmarking Multiple Sequence Alignment Algorithms

Previous benchmarking strategies [39] can be classified into three types: (1) those using simulation, (2) those using statistical methods, and (3) those using expert information. Each of these categories is explained below.

1.5.1 Simulation-based Benchmarking

Simulation-based benchmarking is performed by using a simulation tool to generate a set of sequences combined with their true alignment file, asking a given alignment tool to predict the alignment of those sequences, and then comparing the true and predicted alignment. There are many simulation tools [40, 31, 41, 42]. These tools differ by their capabilities to model the various types of biological mutations involved in a realistic evolutionary model. A comprehensive simulator tool such as the sgEvolverSimulator [43, 44] generates genome length simulated sequences but lacks in modeling translocations and mobile elements evolution. EvolSimulator [45] is another simulator but it uses a simpler model of evolution and has a focus on ecological parameters. The ALF simulator [46] is another option that models genes and neutral DNA simulation.

For this thesis, we use a simulation software called EVOLVER [47]. EVOLVER can simulate full sized, multi chromosome genome length evolution. It has an extensive set of expert-curated model parameters that models DNA sequence evolution with sequence annotations; a gene model; a base-level evolutionary constraint model; chromosome evolution including inter- and intra- chromosomal rearrangements; tandem and segment duplications; and mobile element insertions, movements, and evolutions. The input to EVOLVER is a phylogenetic tree, the root genomic sequence and a set of simulation parameters (rate of

various kinds of mutations). EVOLVER keeps track of the mutations that occur during evolution along a phylogenetic tree. The tool knows the exact homology among the sequences at the leaves of the tree and can produce their true alignment. The leaf sequences are then fed into a MSA tool to obtain another alignment. The two alignments are compared using various methods but the most common one is calculating the specificity (fraction of aligned sites that are homologous) and sensitivity (fraction of homologous sited correctly aligned). The methods used for comparison are statistical methods which are discussed in the next section.

The advantages of simulation based techniques is three fold: (1) the 'true' homology is known, (2) new data can be generated as desired and (3) different scenarios can be modeled during evolution. However, there is a risk of simulated data have different properties than the actual biological data and the alignments being dependent of the evolution parameters.

1.5.2 Statistical Assessment based Benchmarking

Many statistical methods have been developed to assess the quality of MSAs. Most of these methods were developed for protein alignments.

The T-Coffee/TCS Index ([48, 49]) assigns a reliability index (normalized between 0 and 1) to every pair of aligned residues in the target MSA, to every column and to the whole alignment. It computes a library of all pairwise alignments for the given set of sequences and estimates the score of aligning two residues R_i^x (i^{th} residue of sequence x) and R_j^y (j^{th} residue of sequence y) by identifying all intermediate residues R_k^z from a third sequence z that connects R_i^x and R_j^y through the two pairwise alignments: $R_i^x R_k^z$ and $R_k^z R_j^y$. The pairwise reliability score is the sum of scores of all $R_i^x R_j^y$ pairs linked through all possible R_k^z residues, normalized by the maximum score over all possible pair combinations involving R_i^x or/and R_j^y. The reliability score of an alignment column is averaged across all pairwise aligned residue scores within the column. These values are combined to give the whole alignment an alignment reliability score.

Another method called the Heads or Tails (HoT) [50], proposed a simple reliability check for MSAs. The given set of sequences (called the heads set) was reversed to form a tails set and both these sets were aligned independently. The two alignments were com-

pared by calculating the fraction of identical alignment columns and the proportion of residue pairs that are paired identically in the two alignments. HoT measures more the consistency of an alignment algorithm with alternate data, rather than its accuracy [51]. The HoT method was extended [52] to measure the residue-pair reliability, column reliability and alignment reliability. For any given MSA S, a guide-tree was created using the CLUSTALW algorithm. Each internal branch of the tree was used to partition the input sequences into two subsets. For each of these sub sequences two alignments were calculated by CLUSTALW: one for the given sub sequence (heads set) and one for the reversed sub sequences (tails set). This way each internal branch would have eight MSAs associated with it. This process is repeated for each internal branch and the set of alignments generated is termed as the guide tree alignment set, represented as ${}^{gt}AS$. The residue pair reliability score for all residue pairs in S is calculated as the proportion of alignments in ${}^{gt}AS$ that match in S. This residue pair reliability score is averaged over columns and the complete alignment to generate the column reliability and alignment reliability scores respectively.

The GUIDANCE (GUIDe tree based AligNment ConfidencE) [53] tool is another such algorithm that measures the reliability scores (pair-wise, column and complete alignment). For an input MSA, the algorithm first generates multiple inferred phylogenetic guide trees based on bootstrapping technique [54]. For each of the trees, a perturbed MSA is generated using the input sequences and either the MAFFT [55], PRANK [56] or CLUSTALW alignment algorithm. The input MSA is compared to all the generated perturbed MSAs to assign a reliability score to each column, alignment pairs and the complete alignment. The above methods along with some other methods like Zorro [57] and ALiScore [58] were primarily developed for protein alignments with sequences of moderate length.

Prakash et al. [59] developed a method for measuring the accuracy of whole-genome DNA alignments using a statistical method known as StatSigma-w. StatSigma-w is an extension of StatSigma [60], a method that was used to assess if a MSA is contaminated with one or more unrelated sequences. Given a MSA and a phylogenetic tree, StatSigma computes a p-value for each of several null hypothesis cases. Each branch of the phylogenetic tree is related to a null hypothesis. The null hypothesis of a branch k is defined as the branch being 'unrelated', i.e., the sub-alignment corresponding to the left and right subtrees are independent rather than homologous. The assumption is that after rejecting all

null hypotheses, we can infer that all the sequences in the alignment are related. Extending StatSigma to every region of the alignment (StatSigma-w) can detect suspicious alignment regions in the whole chromosome. The authors identified 9.7% (27 Mbp) of the human chromosome I alignment (on the UCSC genome browser) to be suspiciously aligned.

Statistical measures are attractive because they are generally quick and easy to use. However, without having a gold standard to compare them against, these assessments can only serve a proxy to a true assessment of accuracy.

1.5.3 Expert Information based Benchmarking

For protein MSA benchmarking techniques, the $3D$ structural and functional information could assist in determining the true alignment ([61, 62]). However, DNA alignment are much harder to assess because we lack external criterion to assemble expert level benchmarks [63]. DNA alignment are usually evaluated using ad hoc expert information ([64, 65]). The main advantage of using expert information while evaluating alignments is that the objective is clear before assessing the alignment. However, such expert information could only assess part of the alignment (for ex. coding exons), and this method is infeasible for all alignments in general.

The next section is about the Alignathon project [66], which was an online competition to assess whole-genome alignment methods, and was one of the motivating factors for this thesis.

1.6 The Alignathon Project

As in the Assemblathon project [67] that assessed the various assembly methods for sequencing technologies, the Alignathon project was a competitive evaluation of whole genome alignment techniques. Teams submitted their alignments and assessments were performed collectively on the submissions. Three datasets were used for evaluation, two were simulated sets based on primate and mammalian phylogenies, and one comprised of 20 real fly genomes. Ten teams made 35 submissions using 12 different alignment methods. The submissions were assessed independently using simulation and statistical techniques. Both the benchmarking techniques found that there were significant differences in accuracy among

the contemporary alignment tools (MULTIZ, TBA, PSAR-Align [68], VISTA-LAGAN among others). A considerable difference was found in the alignment quality of different annotated regions, and very few tools aligned the duplications. Most of the tools worked well at alignments along short evolutionary distances and very few performed competitively when the sequences were far apart in the phylogenetic tree. It showed that more work is needed to identify likely errors in alignments, and correct them.

1.7 Problem Definition

We saw that whole genome alignment of sequences is a computationally difficult problem and most of the aligners sacrifice on accuracy for a number of reasons. This motivated us to build a tool that can detect errors in the alignments produced. The rest of this thesis deals with solving the problem defined below:
Given: A multiple sequence alignment
Predict: The position and type of errors in the alignment
It is important to note here that the true alignment is not given or known. The goal is not to evaluate an alignment algorithm but to predict suspiciously aligned regions within an alignment. The alignment tool chosen for this thesis is MULTIZ but our approach could be applied to any of the alignment tools.

1.8 Outline of this thesis

The next chapter (Chapter 2) gives a general overview of the machine learning algorithms used in this thesis. Chapter 3 explains the method we developed to solve the problem described in section 1.7. Chapter 4 describes the results obtained, and finally chapter 5 concludes the thesis and describes possible extensions. The work in chapters 3 and 4 were done in collaboration with Ramchalam Kinattinkara Ramakrishnan whose contribution was approximately 20%, although I was the leader of the project, and all the text of my thesis was written by me.

2

Overview of Machine Learning Approaches

Machine learning is the field of computer science that uses statistical methods to give machines the capability to learn from data without being explicitly programmed [69]. Broadly, the field of machine learning can be categorized into three categories:

- **Supervised Learning:** In supervised learning each input example has an associated label. The task of the machine learning algorithm is to learn a function (also called the hypothesis function) that maps the input data to their corresponding labels in order to categorize unseen data into their correct labels. Supervised learning is further subdivided into classification or regression problems. When the labels are categorized into two or more discrete classes, the problem is termed as a classification supervised problem. In regression problems, the labels are not categorical but instead are continuous real numbers.
- **Unsupervised Learning:** Unsupervised learning deals with problems where the input data has no associated labels. Here, the algorithms are designed to learn patterns within the data itself, for example to cluster them in groups or to infer distances between themselves.
- **Reinforcement(Active) Learning (RL):** Reinforcement learning is a form of learning where the goal is to learn a sequence of steps to optimize costs/rewards. The algorithms designed in RL learn the correct actions to take in an environment by tak-

ing random actions initially and improving over time based on the rewards observed after every action.

According to the literature surveyed for this thesis, machine learning has not been applied to detect errors in whole-genome alignments yet. However, scientists have used machine learning algorithms to detect latent errors in computer code [81] or for finding anomalies in the dataset such as for spam detection [82], cyber security intrusion detection [83], etc. It is not easy to draw a direct parallel between these works and ours because they do not deal with finding errors on data that is the output of an algorithm but deal with raw data sets. In this thesis, we deal with a supervised classification problem and the rest of the chapter is focused on this.

2.1 Bias vs Variance

There are two kinds of errors associated with a machine learning model. Bias is the error that measures how far is the true hypothesis from the hypothesis space considered by the predictor. The goal of any algorithm is to reduce the bias as much as possible but in pursuit of this, it is possible that the algorithm learns the training data and does not generalize well for unseen data. This leads to the other kind of error, i.e., variance. Variance measures how large a small change in input data can affect the predictions. Ideally we want our model to have low bias and low variance but there is a trade-off between these two types of errors. Complex models tend to have low bias but high variance and vice-versa. Figure 2.1 depicts this trade off with respect to model capacity (or model complexity) of a machine learning model.

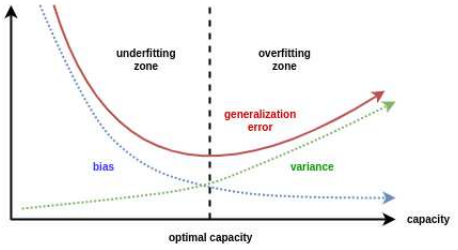


Figure 2.1: Bias variance tradeoff as a function of model capacity. Figure is taken from [70].

Underfitting occurs when the model has still not learnt patterns in the data and it suffers from high bias. Conversely, overfitting occurs when the model memorizes the labels associated with the training data (implying very low bias) but doesn't generalize well to unseen data (because of high variance).

2.2 Supervised Classification Problem

In supervised classification problem, each input example has an attached label and our goal is to learn a function that maps the data to its corresponding label. Formally, given a data set D that consists of training examples, each example is represented by
$x_i =< x_{i1}, x_{i2}, x_{i3}, ..., x_{in}, y_i >$, where

- $x_{ij} \epsilon X_j$ is the value of the j^{th} feature of the i^{th} example
- $y_i \epsilon Y$ is the label for the i^{th} training example, where Y is a finite discrete set

The goal is to learn a function mapping $f : X_1 \times X_2 \times X_3 \times .... \times X_n \rightarrow Y$. The function f is also called the $hypothesis$ function. There are many algorithms to find a good hypothesis function but in this thesis, we apply two such algorithms: random forest classifier and artificial neural networks.

2.2.1 Random Forest Classifier

Before describing a random forest classifier, we need to understand the decision tree classifier. A decision tree constructs a tree like structure where each node makes a decision based on a test of one of the features (or on combination of features) and splits the data based on the outcome of the tests. Generally, this is done till the leaf node contains instances of a single class. Any unseen data is classified by traversing the path to the leaf along the tree based on the outcome of the attributes of the data at each node. A simple decision tree is shown in Figure 2.2 where there are two output classes - positive (P) and negative (N), and the goal is to predict if a specific user performs an activity (for example playing golf) on Saturday mornings.

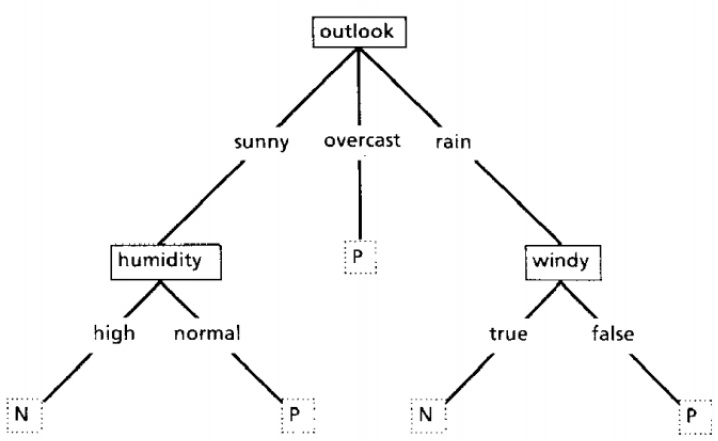


Figure 2.2: A simple decision tree. Figure is taken from [71].

The algorithm to construct a decision tree from input data is as follows:

Algorithm 1 Training a Decision Tree

Given a set of labeled training instances
1: **if** all the training instances have the same class **then**
2: Create a leaf with that class label and exit
3: **else**
4: Pick the best test to split the data on
5: Split the training set according to the values of the outcome of the test
6: Recursively repeat steps 1-5 on each subset of the training data
7: **end if**

Choosing the best test to split the data on each node (step 4 of the algorithm above) is done by finding the information gain [72] for each test and picking the one with the highest information gain. Decision trees are prone to over-fitting because in the process of making the leaves pure the tree may not predict well for test data.

A random forest classifier overcomes this problem of over-fitting by reducing variance. A random forest is a collection of many decision trees where each split in the decision tree is built by choosing one feature out of a random subset of features from the entire feature set. Also each decision tree is trained on a randomly resampled training set (with repetitions). Unseen data is labeled by picking the mode of the classes predicted by all

of the decision trees. Training a random forest classifier implies training many decision trees. The number of decision trees, the number of features to be selected for each split, the depth of each tree are some of the important hyper-parameters of this classifier. Generally, random forest classifier trains faster than other classification algorithms. A disadvantage of using random forest classifier is that they cannot be easily applied to train large datasets that cannot be stored completely in the computer's memory.

2.2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are a network of simple computing units that can collectively learn a complex function. Neural Networks are efficient in learning complex functions and can provide good accuracies for complex tasks such as text-related problems [77], time series data analysis [78], etc. In bioinformatics, neural networks have been used successfully in problems such as predicting DNA binding sites [79], prediction of cancer [80], etc. To understand how ANNs work, first we need to understand the functioning of a perceptron classifier. A perceptron, shown in Figure 2.3, is a classifier used for supervised binary (two output classes) classification.

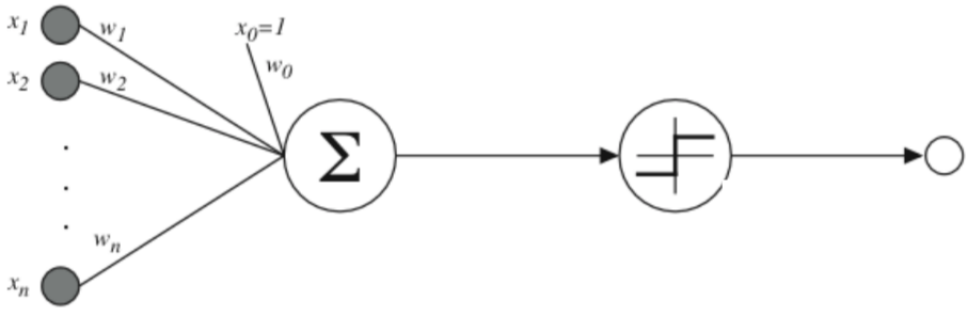


Figure 2.3: A Perceptron classifier. $x_1, x_2, .., x_n$ is one training vector. x_0 is called a bias input. w_i is the weight associated with each x_i. The first computing cell takes a linear combination of input. The second is a signum function. The last is the output cell.

It takes as input $x_1, x_2, ..., x_n$, computes its weighted sum (w_i is the weight associated with each x_i) and passes this sum to another function called as $activation function$, which thresholds the input to classify them to one of the two classes. The activation function in

the perceptron shown in figure 2.3 is a signum function. Using the signum function, the example could be classified as belonging to either class +1 or -1, based on the equation:

$$h_{\mathbf{w}}(\mathbf{x}) = sgn(\mathbf{x} \cdot \mathbf{w}) = \begin{cases} +1, & \text{if } \mathbf{x} \cdot \mathbf{w} > 0 \\ -1, & \text{otherwise} \end{cases}$$

In the training phase, the algorithm adjusts the weights based on the input training data. After initializing the weights randomly (to small numbers or zeros), the classifier predicts the output for each input sample, compares it to the actual output and based on the difference (measured by a $lossfunction$) adjusts the weight accordingly. Once the weights are trained, any unseen data could be classified by taking a linear combination of the test data with the trained weights and calculating its signum. The main drawback of using a single perceptron classifier is that it cannot learn a non linear boundary to separate the two classes (for example an XOR function). The hypotheses space of many complex problems is non linear. To overcome this we connect many perceptron units to form a network, this network is called an artificial neural network (simply referred as neural network) [73]. Figure 2.4 shows a neural network with three layers.

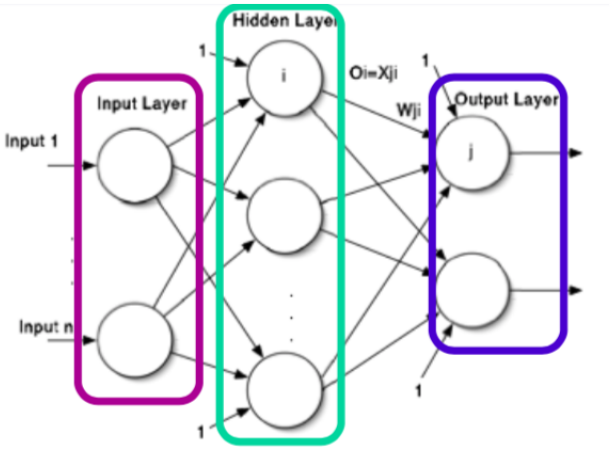


Figure 2.4: Neural network with three layers. Layer 0 is the input layer. Layer 1 is the hidden layer. Layer 2 is the output layer. w_{ji} denotes the weight on connection from unit i to unit j. Output of unit i, denoted o_i is the input to unit j denoted as x_{ji}. In this figure, the weights between input layer and hidden layer are not shown for simplicity.

2.2 Supervised Classification Problem

Generally, there can be K layers in a neural network. Layer 0 is called the input layer, it simply copies the input. Layer $1, .., K-1$ are called the hidden layers. The last layer K is called the output layer. Each neuron in layers $1, .., K$ performs a weighted sum of its input and applies an activation function to the sum. The network is also called a feed forward neural network because the outputs of unit in layer k are inputs to the units in layer $k+1$. There are no connections among units in the same layer, and there are no connections going back to a previous layer. You could also observe that all the units of layer k are connected to all the units of layer $k+1$ which implies that this is a fully connected feed forward neural network.

Similar to a perceptron, in the training phase, the learning algorithm adjusts all the weights in the network based on the input data. After initializing the weights randomly, the algorithm computes the output for each training example. Based on the difference (estimated by a loss function) between the true output and predicted output the weights are adjusted by an optimization technique. There are many optimization techniques but the most commonly used is gradient descent. Gradient descent adjusts the weight by calculating the gradient of the loss function starting from the output layer and back-propagating the gradient across each hidden layer. During the back propagation step it calculates the amount of correction required in all the weights of the network. The weights are adjusted at each iteration of the training phase. The training approach is summarized in the below steps:

- Initialize all weights to small random numbers
- Repeat until convergence
  - Pick a subset X of the training examples randomly
  - Feed the example through the network to compute output at the output unit
  - For the output unit, compute the correction by calculating the gradient of loss function
  - For each hidden unit h, compute its share of correction by back propagation
  - Update each network weight by gradient descent with the equation below. $w_{h,i}$ is the weight of unit i in layer h. α is the $learning\ rate$. $\delta_h x_{h,i}$ is the gradient

at unit i of layer h.

$$w_{h,i} \leftarrow w_{h,i} + \alpha \cdot \delta_h \cdot x_{h,i}$$

Depending on how the subset X of examples is selected at each iteration, we obtain three variants of gradient descent

- **Batch Gradient descent (BGD)**: X is the entire set of training examples.
- **Stochastic Gradient descent (SGD)**: X consist of a single example at a time.
- **Mini batch gradient descent (MGD)**: X is a set of fixed number of randomly selected examples.

BGD is very slow for large datasets and it requires the entire dataset to be stored in the computer's memory, which makes it infeasible for large datasets. This is overcome by using SGD. SGD is much faster than BGD but it suffers with high variance while setting the weights due of which it may overshoot the mimima of the hypothesis function. However, with variance decreasing techniques it is shown that SGD can converge to the same optima as BGD [74]. MGD takes the best of both worlds and performs an update for every mini batch of the training data. This way there is less variance in updating the weights and the convergence is much more stable. Some advanced optimization techniques such as the Adam optimizer [75] use a technique called momentum that accelerates the gradient descent in the direction of optima by adding a fraction of the update vector from the previous step to the current update vector. The other advanced techniques, such as Adagrad [76] modify the learning rate in the training phase. For parameters associated with frequently occurring features, a smaller learning rate is used compared to the parameters associated with less frequently occurring features.

It may take a lot of time and many experiments to tune the large number of hyper parameters to get satisfactory results. To tune the hyper parameter values, the complete data is generally divided into three disjoint sets: training set, validation set and test set. For each possible value of the hyper parameter the model is trained on the training set and its accuracy is obtained on the validation set. The hyper parameter with the best accuracy on the validation set is applied on the test set.

2.3 Evaluating Model Performance

There are many metrics to measure the correctness of a trained machine learning model. The section below describes the relevant performance metrics for this thesis.

2.3.1 Confusion Matrix

For classification problems the confusion matrix helps us to visualize the performance of the classifier. The rows represent instances of the actual class while the columns represent instances of the predicted class or vice-versa. The value at row r and column c of the confusion matrix represents the number of instances belonging to class r but predicted to belong to class c. If there are n classes this would be a $n \times n$ matrix. This helps to visualize the extent by which the model is confused among the classes. For a classification problem with two classes the 2×2 matrix is shown in Figure 2.5.

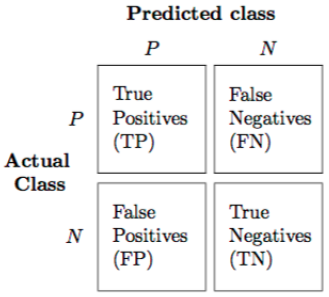


Figure 2.5: 2×2 confusion matrix.

For actual class P, the number of instances predicted as P are True Positives (TP) while the number of instances predicted as N are False Negatives (FN). For actual class N, the number of instances predicted as P are False Positives (FP) while the number of instances predicted as N are True Negatives (TN). It is important to classify these errors separately because in some problems, for example in detecting the recurrence of cancer, false negatives are more serious than false positives. For both the classes, accuracy can be easily derived from the confusion matrix. The accuracy for class P (commonly referred to as sensitivity) is $\frac{TP}{TP+FN}$ and the accuracy for class N (commonly referred to as specificity) is $\frac{TN}{TN+FP}$. Similarly, we could also calculate accuracy for each class in larger confusion ma-

trices. The False Positive Rate (FPR) is calculated as $\frac{FP}{FP+TN}$ and the False Negative Rate (FNR) is $\frac{FN}{FN+TP}$. In the detection of recurrence of cancer we would like FNR to be as low as possible but wouldn't worry as much about FPR.

To conclude, this chapter gave a brief overview of the machine learning methods used in this thesis. The next chapter explains the data generation and feature engineering methods developed to solve our problem.

3

Methods

As discussed in Section 1.5, there are many ways to find errors in multiple sequence alignments. To use machine learning techniques to detect errors in an alignment we need to feed into our model examples of correct and incorrect alignments. Simulation based tools can generate a true alignment. Other benchmarking approaches discussed previously such as using expert information or statistical techniques cannot be incorporated into a machine learning problem. Hence, our approach is one that relies on simulation-based techniques. We decided on a small set of species and assume that the set of species and its corresponding phylogenetic tree is given to us as input (explained later in section 3.1). The approach could be explained in the steps below, each of which is detailed in subsequent sections.

1. Use a simulation tool, which takes as input the genomic sequence of the root of the phylogenetic tree, evolves the sequences along the branches of the phylogenetic tree and outputs sequences of the leaf species and their MSA.This alignment is referred to as the *true alignment* from now on. The simulation tool must take into consideration the various types and rates of evolutionary events (substitutions, indels, duplications etc.).
2. Generate the MSA of the sequences at the leaves of the tree using an alignment tool. From now on this is referred to as the *predicted alignment*.
3. Compare the alignment files generated by step 2 and step 3 to identify portions of the alignment where the *predicted alignment* agrees with the *true alignment*, and those where they disagree.

4. Using the data produced in step 3, train a machine learning model to identify the portions of the predicted alignment that are incorrect.

The below sections will elaborate on each step mentioned above.

3.1 Species Selection

We decided to perform our analyses on a set of 10 mammalian species (Figure 3.1) that represent a good sampling of the diversity in this phylum. It could be assumed that the results on the UCSC 100-way vertebrate genome phylogenetic tree would be similar to what we get on this smaller phylogenetic tree. The topology and the branch lengths of the below tree is deduced from the Newick format of the UCSC 100-way vertebrate genome phylogenetic tree [84].

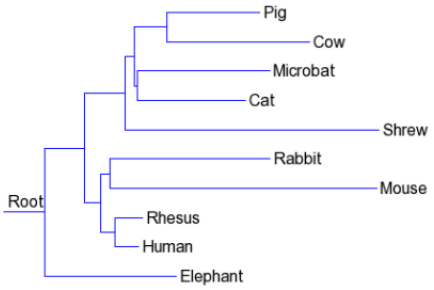


Figure 3.1: Phylogenetic tree of the 10 selected species.

The tree in Newick format can be written as (((((Cat:0.145927, Microbat:0.17896) C-M:0.003836, (Cow:0.191657, Pig:0.124472) C-P:0.044324) C-M-C-P:0.012255, Shrew:0.341864) C-M-C-P-S:0.0232,((Human:0.032505, Rhesus:0.037831) H-R:0.018791,(Mouse:0.360046, Rabbit:0.214997) M-R:0.013028)H-R-M-R:0.021174) C-M-C-P-S-H-R-M-R:0.022494, Elephant:0.176934).

3.2 Data Generation

3.2.1 Generating the True Alignment

As discussed in Chapter 1, a comprehensive suite of tools, EVOLVER [47] (used in the Alignathon project [66]) was developed to generate simulated genomic sequences. This tool considers various intrachromosomal and interchromosomal biological processes such as substitutions, insertions, deletions, duplications, translocations and inversions while simulating the evolution of these sequences. The expert curated parameter set that includes the rate of occurrence of these biological events with respect to the length of the input sequence was presented along with the simulation tool. EVOLVER simulates sequences along a single branch of the tree. To generate simulated sequences based on a phylogenetic tree we need to run EVOLVER for multiple branches. The tools required to simulate sequences along a phylogenetic tree were developed in the Alignathon project [66]. These tools called EvolverSimControl available at https://github.com/dentearl/evolverSimControl, EvolverInfileGeneration available at https://github.com/dentearl/evolverInfileGeneration/, along with complementary tools such as mafJoin available at https://github.com/dentearl/mafJoin/ can construct Multiple Alignment Format (MAF) files containing the entire multiple sequence alignment between sequences of leaves, internal nodes and roots of a phylogenetic tree.

The simulation was started by using a 10 Million base pair (Mbp) subset of chromosome 22 from the well annotated human genome version 18 (hg18) of the UCSC browser. This 10 Mbp sequence was used as the ancestor sequence (root) of the phylogenetic tree shown in Figure 3.1. Using the actual human DNA sequence as our root sequence appears counterintuitive at first because human is one of the leaves in our tree, but it makes the root sequence realistic to an actual genomic sequence (in terms of DNA sequence properties such as percentage of GC content, etc.) which is preferable than using a random sequence for the root.

At the end of the simulation, along with the MAF file we also get the fasta files containing the evolved genomic sequence at each node of the tree. The length of sequences at the leaves of the tree varied from 11 Mbp to 13 Mbp. It took approximately 36 hours on a machine with 64 GB of RAM and 12 CPU cores, to generate the simulated data. Since

the simulation tool keeps track of the history of each nucleotide during the entire evolution and uses it to align them correctly, we can consider this MAF file as the *true alignment* or the *correct alignment*.

3.2.2 Generating the Predicted Alignment

We can utilize the fasta sequences of the leaf species generated by the simulation tool, and feed them into any alignment tool to generate a predicted MAF file. The alignment tool that we have chosen for this thesis is MULTIZ [32]. It is one of the most popular and widely used alignment tool for whole-genome alignments. It is also used for the alignments shown in the UCSC genome browser [35]. Before feeding the fasta files to MULTIZ, each sequence was fed into a masking tool called RepeatMasker [85]. RepeatMasker identifies and soft-masks repetitive elements in the sequence, which helps to speed up the running time of MULTIZ. All the species were aligned pair wise with human as the reference. To generate pairwise alignments for MULTIZ, LASTZ was run with the following flags: –step=10, –gapped, –nochain, –gfextend and –strand=both. MULTIZ combines these pairwise alignments to generate an alignment file containing all the ten species. This MAF file would be considered as the *predicted alignment*.

3.3 Comparing True and Predicted Alignment

After obtaining the true and predicted alignment files, our task is to compare them. We can compare alignments by comparing the index positions of the nucleotides that are aligned together. For example, refer to the small alignment blocks shown in Figure 3.2 among three species: human, microbat and shrew.

3.3 Comparing True and Predicted Alignment

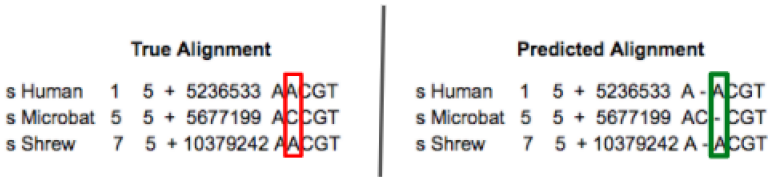


Figure 3.2: True vs predicted alignment for a small alignment block. The first number after the species name represents the starting index position of the first nucleotide. The next number represents the number of nucleotides (excluding gaps) present in the alignment block for that species. The alignment is from a positive or negative strand, which is represented by + or -. The last number represents the total size of the sequence of that species.

Referring to the alignment column highlighted in red in the true alignment, the nucleotide at index position 2 of human is aligned to the nucleotide at index position 6 of microbat and to the nucleotide at index position 8 of shrew. Similarly, in the alignment column highlighted in green in the predicted alignment, the nucleotide at index position 2 of human is aligned to a gap in microbat and to the nucleotide at index position 8 in shrew.

If we strictly compare all index positions of the species in the true alignment column to that of the predicted alignment column, then we would annotate this predicted alignment column as an incorrect alignment because $[2, 6, 8] \neq [2, \phi, 8]$. Understandably, this is a very strict measure of comparing the two MAF files. To make a finer comparison between the true and predicted alignments, we compare the indices pairwise with respect to the reference sequence (here, human). For example, in the alignment shown in Figure 3.2, we observe that the index position 2 of human is aligned to the index position 8 of shrew in both the true and predicted alignments. Thus for this alignment column, we would annotate the Human-Shrew pair as a correct alignment. This would be repeated for all the other species with respect to human. If we have n species we would have $n - 1$ annotations for each alignment column. Based on whether a nucleotide is aligned to another nucleotide or if its aligned to a gap at the index positions for each pair of Human-Non Human species there is a total of five different possible labels. Refer Table 3.1, where the alignments are represented using the index positions.

3.3 Comparing True and Predicted Alignment

	True Alignment	Predicted Alignment	Class Description	Class Label
Human	1	1	Correct	0
Non Human	2	2	Match (CM)	
Human	1	1	Correct	1
Non Human	–	–	Gap (CG)	
Human	1	1	Over	2
Non Human	–	2	Alignment (OA)	
Human	1	1	Under	3
Non Human	2	–	Alignment (UA)	
Human	1	1	Mis	4
Non Human	2	3	Alignment (MA)	

Table 3.1: The five possible classes of each human-non human pair. The numbers represent index positions in the DNA sequence of the species. '-' represents gap in the alignment. For example, if index position 1 of human is aligned to a - in the other species in the true alignment but in predicted alignment index 1 of human is not aligned to a - but is aligned to any nucleotide then that alignment pair would be labeled as over alignment.

Labeling the data this way would make it possible to not only find the erroneous alignment columns but also tell the type of error that occurs in each species.

To summarize, we compare each alignment column of the predicted alignment to the corresponding column of the true alignment. Corresponding columns here imply that they both have the same human index position in the alignment. Columns where there is a gap for human are ignored. Since we have 10 species in our phylogenetic tree, we will have nine labels for each alignment column and each of these labels could belong to one of the five classes described in Table 3.1.

3.4 Feature Enhancement

The alignment contains characters (A,C,G,T,-). To feed these characters into a machine learning model we have to encode these characters numerically. These characters are one-hot encoded as shown in Table 3.2 before being fed into a machine learning model.

Nucleotide	Encoding				
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	1

Table 3.2: One-hot encoding of nucleotides.

Along with the nucleotides, we can feed in additional features that may help the classifier to find erroneous alignments. We consider four different types of features here and each one is explained below.

3.4.1 Neighboring Alignment Columns

The goal of an alignment algorithm is to align conserved sequences together and these conserved regions generally occur in consecutive chunks of 10 to 1000 nucleotides. Also, in an alignment of a large genomic sequence with another large sequence, a mutation in one alignment column will affect the neighboring alignments as well. Due to this, an error in aligning one column would often result in errors in the neighboring columns as well. Therefore, in addition to the features encoding the alignment column whose correctness we want to predict, we will feed in the encoding of neighboring alignment columns on both sides. As a result, the classifier can look at an alignment block instead of a single alignment column to predict the class labels of the central alignment column. The exact number of neighboring columns to be used will be set experimentally. Figure 3.3 shows an alignment block where we take three neighboring columns on each side of the central

alignment column, totaling up to seven alignment columns.

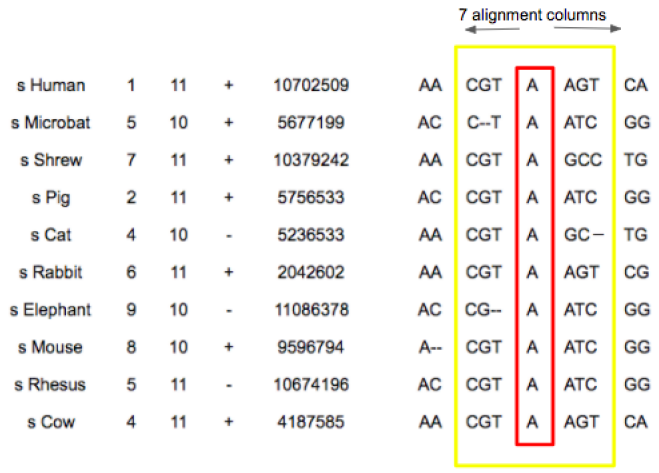


Figure 3.3: Feature set- neighboring alignment columns.

3.4.2 Match/Mismatch bits

As seen in Table 3.1, the class labels were generated after comparing the nucleotide of every species to the nucleotide of human. If the nucleotides of the other nine species match to the human nucleotide in an alignment column (and neighboring columns) then it is more probable for the labels of that column to belong to the “Correct Match" class. Therefore we encode this information in a single bit per species. For each pair of Human - Non Human species we encode this feature as 1 if the nucleotides match, 0 otherwise.

For example, in Figure 3.4 for the alignment column consisting of ten species, the match mismatch feature vector of length nine is shown.

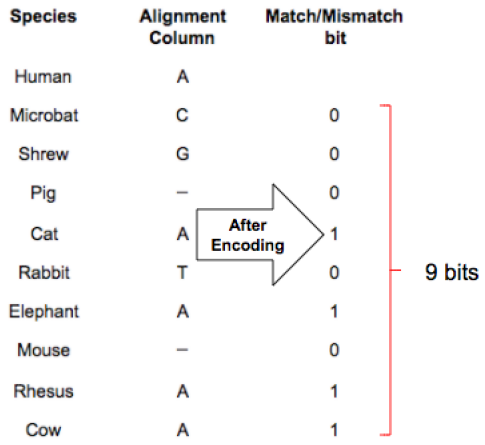


Figure 3.4: Feature set- match/mismatch bits.

3.4.3 Ancestral Nucleotides

In section 1.2 we saw that sequences evolve over time along the branches of a phylogenetic tree. During evolution these sequences undergo various mutations such as substitutions, insertions, deletions, duplication, translocations and inversions, among others. If we could know the exact changes in the sequences along a phylogenetic tree we could deduce the correct alignment of the descendants by taking into account these changes. Hence, if we know the ancestor nucleotides at each internal node of the phylogenetic tree, and feed this information to a classifier, it may help the classifier in predicting the class labels.

Section 3.2.2 explains how the predicted alignment was generated using MULTIZ. However, we do not have the sequence of the ancestors to obtain an alignment with them. Fortunately, there are some tools that take an existing alignment as input along with the phylogenetic tree and produce an output alignment containing the alignments among all nodes of the phylogenetic tree. The output alignment file contains the alignment of the sequences at the leaves of the tree along with the inferred ancestral sequences as well.

Ancestor 1.0 [86] is one such tool that predicts the ancestor nucleotides and outputs an alignment along with the ancestors. For each alignment column the nucleotides (or gaps) predicted at the internal nodes of the tree makes up this feature set.

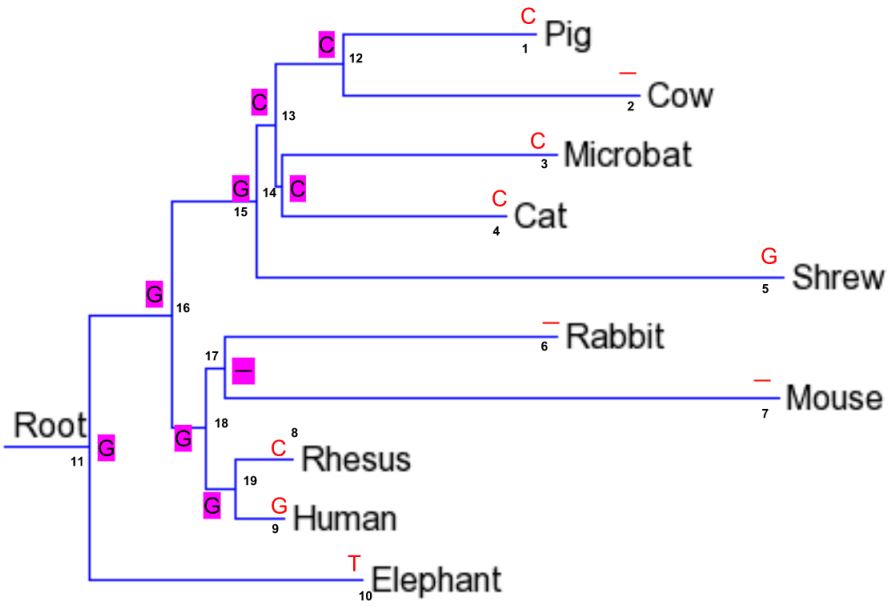


Figure 3.5: Feature set- ancestor nucleotides.

Figure 3.5 shows the nucleotides of an alignment column (in red) along the leaves in the tree. There are nine internal nodes in the tree marked with numbers 11 to 20. Ancestor 1.0 generates the nucleotides at each of these nodes (highlighted in magenta). As part of this feature set these nucleotides are fed to the classifier after encoding them with one-hot encoding (See Table 3.2). This results in $9 \times 5 = 45$ additional binary features.

3.4.4 Mutations along each branch of the phylogenetic tree

Along with the nucleotides or gaps at each internal node of the tree, we could explicitly encode the type of mutation occurring along each branch of the phylogenetic tree. The mutation could be categorized into either a substitution or an indel. Encoding each kind of substitution in the same manner is inaccurate as substitutions can further be categorized into two types: transitions and transversions. Transitions are substitutions of the type $A \leftrightarrow G$ or $C \leftrightarrow T$. The other mutations, namely $A \leftrightarrow C$; $A \leftrightarrow T$; $C \leftrightarrow G$; $G \leftrightarrow T$ are called as transversions. Transitions are known to be more frequent than transversions. With substitutions, insertions or deletions should also be considered. Traversing from an ancestor node to a descendant node, insertion implies the addition of a nucleotide along the branch

while deletion means removal of a nucleotide. It is also possible that there is no mutation or indels along a branch and the character of an ancestor (nucleotide or gap) remains the same for the descendant. However, it is important to distinguish a nucleotide not changing over a branch from a gap not changing over a branch. We encode the above details using six bits of information per branch of the phylogenetic tree after comparing the character of the ancestor with that of the descendant in that branch.

The six bits are:

- **Transition Bit:** Set to 1 if there is a mutation of type transition, else to 0.
- **Transversion Bit:** Set to 1 if there is a mutation of type transversion, else to 0.
- **Insertion Bit:** Set to 1 if an insertion occurs along that branch, else to 0.
- **Deletion Bit:** Set to 1 if a deletion occurs along that branch, else to 0.
- **Nucleotide Match Bit:** Set to 1 if both ancestor and descendant are the same nucleotides, else to 0.
- **Gap Match Bit:** Set to 1 if both ancestor and descendant are gaps, else to 0.

The Table 3.3, shows few different encodings for different branches of the tree shown in Figure 3.5. Please refer to the node numbers from the Figure 3.5.

Ancestor Node No.	Descendant Node No.	Ancestor Nucleotide	Descendant Nucleotide	Transition	Transversion	Nucleotide Match	Deletion	Insertion	Gap Match
12	1	C	C	0	0	1	0	0	0
17	7	$-$	$-$	0	0	0	0	0	1
11	10	G	T	0	1	0	0	0	0
18	17	G	$-$	0	0	0	1	0	0

Table 3.3: Encoding examples for mutations along a branch.

There are 18 branches in the tree and encoding six bits of information per branch generates 108 bits of features per alignment column.

3.5 Class Imbalance

After labeling the different classes for all the nine human-non human pairs of species in the entire dataset it was observed that there was a class imbalance among the five classes (Refer Figure 3.6).

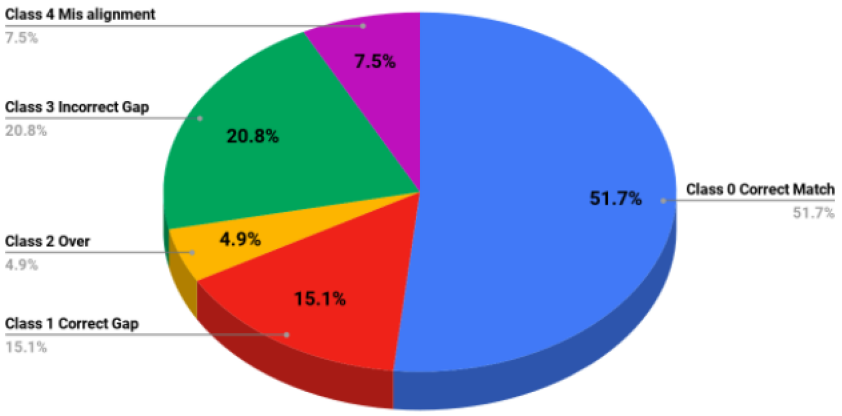


Figure 3.6: Class distribution of the five classes for all the nine pairs of species.

The predicted alignment agreed with the true alignment 66.8% of the time, which is the sum of class 0 (CM) and class 1 (CG). The species-wise class distribution is given in Table 3.4.

3.5 Class Imbalance

Species	Correct Match	Correct Gap	Over Alignment	Under Alignment	Misalignment
Cow	40.076%	19.257%	9.142%	23.82%	7.706%
Cat	53.187%	14.717%	8.718%	15.56%	7.819%
Mouse	26.723%	24.444%	8.155%	34.087%	6.592%
Pig	50.141%	15.821%	9.012%	17.561%	7.464%
Rhesus	78.394%	5.488%	5.678%	6.136%	4.303%
Microbat	49.576%	16.676%	8.914%	17.777%	7.056%
Elephant	51.389%	15.801%	8.707%	16.754%	7.349%
Rabbit	50.267%	15.603%	9.037%	16.402%	8.69%
Shrew	24.88%	24.961%	8.156%	35.899%	6.105%

Table 3.4: Class distribution per species.

The species closest to human in the phylogenetic tree, like rhesus, have the highest percentage of "Correct Match" as expected, while the species farthest to human, i.e., Shrew, has the lowest percentage of "Correct Match" but the highest percentage of "Under-Alignment". On one hand it was encouraging to see that MULTIZ did a satisfactory job in aligning these sequences, but on the other hand this made our task of finding errors in the predicted alignment more difficult because any classifier could achieve a decent accuracy by predicting the majority class. Since we have a large dataset with $\sim$ ten million instances, we re-sampled our dataset to delete instances from the over represented class to make all the classes equal in frequency.

For each alignment column we have nine labels, because of which we cannot re-sample the dataset in a way that ensures a uniform class distribution for all species simultaneously. Hence, we decided to focus our analyses only on the Human-Cow pair of species. After re-sampling, the size of the dataset decreased from 10471799 records to 4034749 records. The class distribution of all the species with these ~ 4 million records is shown in Table 3.5.

Species	Correct Match	Correct Gap	Over Alignment	Under Alignment	Misalignment
Cow	20.0%	20.0%	20.0%	20.0%	20.0%
Cat	44.846%	16.363%	14.545%	15.065%	9.18%
Mouse	22.405%	25.695%	12.717%	32.09%	7.093%
Pig	40.571%	17.637%	15.313%	16.919%	9.56%
Rhesus	71.657%	7.303%	9.322%	6.235%	5.483%
Microbat	41.63%	18.172%	14.675%	17.157%	8.366%
Elephant	44.187%	17.703%	14.079%	16.063%	7.969%
Rabbit	44.098%	16.467%	14.038%	15.857%	9.541%
Shrew	20.142%	26.811%	13.064%	33.566%	6.416%

Table 3.5: Class distribution per species after resampling the dataset.

The alignment errors largely occur independently from one species to another, thus resampling did not lead to large changes in the class frequencies for other eight species. The evaluation of our machine learning model is done only on the accuracy for the human-cow label. If we want to predict the label for all ten species then we will need to build ten different models, one for each species. Even though our focus is to build a classifier that could predict the classes for human-cow, it was decided that the dataset should contain the features and labels for all the species because we think there is some correlation among the data that would help our classifier to predict the labels of human-cow alignment pair.

3.6 Machine Learning Models

This section explains the implementation details of the machine learning models.

3.6.1 Artificial Neural Networks

An artificial neural network (ANN) has large number of hyperparameters such as the number of hidden layers, the number of neurons in each layer, the activation function, etc. The exact value of these hyperparameters depends on the kind of problem and data set, and finding these exact hyperparameters over the entire space of hyperparameters is often time

consuming and difficult. A reasonable sized network in terms of number of hidden layers and number of neurons per hidden layer was chosen initially. The architecture is shown in Figure 3.7. We tried to add one more layer to the network but that did not affect the results, so we reverted to the architecture shown in the figure below.

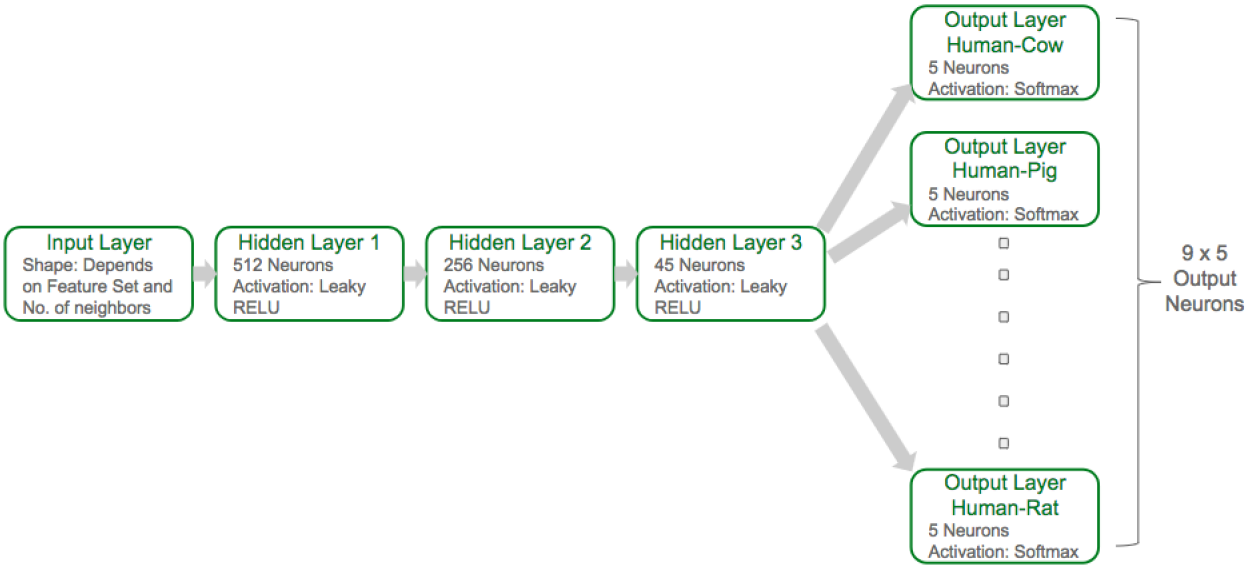


Figure 3.7: Artificial neural network architecture. All the neurons of one layer are connected to all the neurons of the next layer. The bias unit for each hidden layer is not shown in this figure.

For the hidden layers, the activation function leaky RELU was used. Leaky RELU is a modified version of the RELU activation function. It is defined as

$$f(x) = \begin{cases} x, & \text{if } x \geqslant 0 \\ \alpha * x, & \text{if } x < 0 \end{cases}$$

α is generally a small number (~ 0.01). The advantage of the leaky RELU over the standard RELU function (where $\alpha = 0$) is that it never outputs exactly zero for any input thereby not making any neuron inactive in the network.

For each of the nine output labels, the neural network predicts the probability of belong-

ing to the five output classes: correct match, correct gap, over alignment, under alignment and mis-alignment (even though the analysis is done only on the human-cow output pair). The sum of these five probabilities is one. The class with the highest probability is the predicted class for each output layer. ANN was implement using tensorflow [87] and keras [88] machine learning packages.

3.6.2 Random Forests

Random Forests (RF) was implemented using the scikit-learn package [89].

All the experiments with RF and ANN were run on a machine with 64 GB RAM, 12 CPU cores and two Nvidia GPUs. The code is available for reference at `https://github.com/jaspal1329/MSA-ED`.

In conclusion, in this chapter, we discussed about the data generation process and represented our problem as a supervised machine learning classification problem. Some feature enhancement techniques were also proposed along with the implementation details of the machine learning models. The next chapter presents the results obtained using our machine learning model.

4

Results

This chapter discusses the results obtained with the prediction approaches described in the previous chapter. Section 4.1 and section 4.2 explains the results obtained with Random Forests (RF) and Artificial Neural Network (ANN) respectively. Section 4.3 compares the two classifiers. Finally, section 4.4 shows the results obtained when the best learned machine learning model is applied on real alignments.

After generating the data by the methods described in chapter 3, our job is to train a machine learning model that can label each position of the Human-Cow predicted alignment pair to belong to one of the five class labels: Correct Match (CM), Correct Gap (CG), Over Alignment (OA), Under Alignment (UA) and Mis-Alignment (MA). Table 4.1 shows the four different feature sets introduced in the previous chapter. For each feature set we will experiment by varying the number of neighboring alignment columns (Section 3.4.1).

Feature Set	Description
Feature set 0	One-hot encoded nucleotides of the ten species
Feature set 1	Feature set 0 + Match/mismatch bits
Feature set 2	Feature set 1 + Ancestor nucleotides
Feature set 3	Feature set 2 + Mutation along each branch of the phylogenetic tree

Table 4.1: List of different feature sets.

The total number of examples was approximately 4 million (4021490 to be exact). This was split into 70% training data (~ 2.8 million records) and 30% test data (~ 1.2 million records). After training the model using the training data, the accuracy of the classifier was measured on the test data.

4.1 Random Forest Results

The default hyper-parameters of the random forest classifier in the scikit-learn package [89] such as the number of trees, the number of features to consider for each split, the

splitting criteria, etc. work well for most of the classification problems. However, it was observed that if the leaves of all the decision trees in RF were expanded till each leaf was pure (which was the default option), the RF classifier was overfitting. To overcome this the maximum depth of each tree in the RF was tuned by plotting the train and test accuracy for multiple settings of this hyper-parameter. Figure 4.1 shows this curve for the experiment run on feature set 0 with zero neighbors. The classifier overfits as the maximum tree depth parameter is increased. The point in the graph where the test accuracy is highest decides the optimum value of this parameter. The optimal depth in Figure 4.1 is 16 where the overall test accuracy of the classifier is 50.65%.

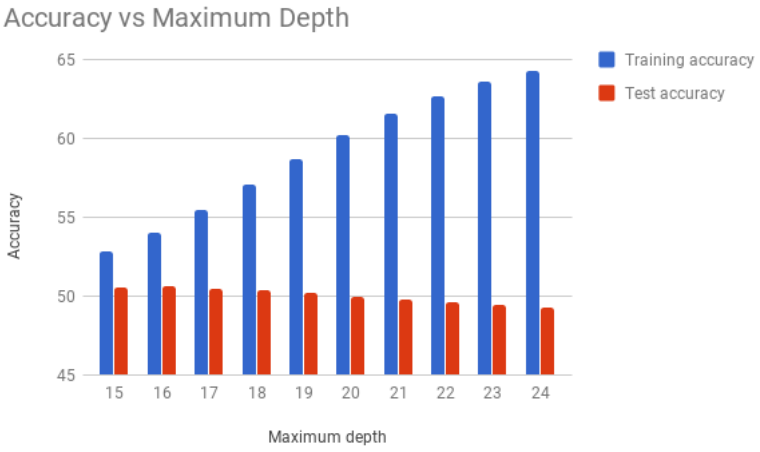


Figure 4.1: Training and test accuracy vs maximum depth. The maximum depth of each tree in the RF is plotted along the horizontal axis. The accuracies are plotted along the vertical axis. As the depth increases the training accuracy keeps on increasing but after an optimum depth the test accuracy starts to decrease. The optimum depth in this case is 16 where the test accuracy is the highest.

In a similar fashion the optimal depth was found for each of the experiments run with RF. This resulted in running many iterations of the same experiment with different maximum depths and it increased the running time of the experiments. While we are not going to report the optimal tree depth for each dataset, we observe that it consistently ranges between 15 and 25.

4.1.1 Feature Set 0

Table 4.2 shows the test set accuracies for feature set 0 as the number of neighbors are increased. In all the results, the number of neighbors shown in the table is the sum of neighbors on both side of the primary alignment column. So 20 neighbors imply 10 alignment columns on each side of the primary alignment column. The accuracy per class is the number of instances predicted for that class divided by the total number of instances belonging to that class.

No. of Neighbors	**Overall Accuracy**	**Class 0 (CM) Accuracy**	**Class 1 (CG) Accuracy**	**Class 2 (OA) Accuracy**	**Class 3 (UA) Accuracy**	**Class 4 (MA) Accuracy**
0	50.65%	54.55%	55.62%	42.67%	59.13%	41.25%
20	52.87%	60.44%	56.58%	44.34%	63.00%	39.96%
40	54.53%	61.45%	58.83%	46.28%	63.87%	42.06%
60	55.68%	61.05%	60.23%	49.42%	64.43%	43.16%
80	56.80%	61.93%	60.21%	51.05%	65.76%	44.88%

Table 4.2: Random forest results: feature set 0. CM, CG, OA, UA, MA stand for correct match, correct gap, over alignment, under alignment, and mis-alignment respectively. All the accuracies presented here are test set accuracies.

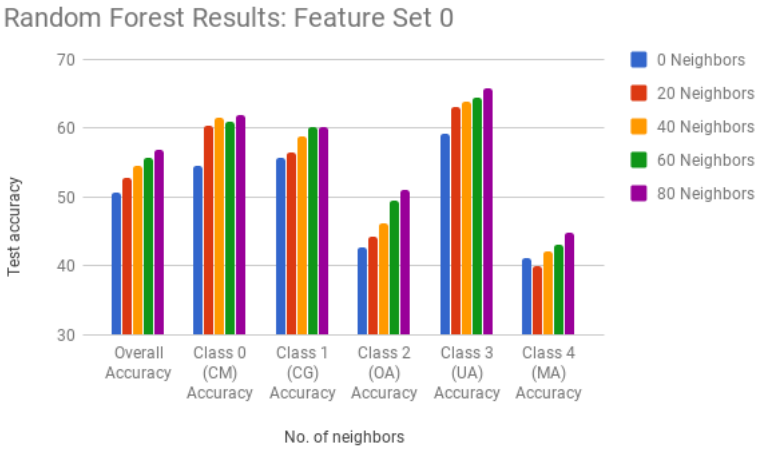


Figure 4.2: No. of neighbors vs test accuracy for data in Table 4.2. The plot shows the change in accuracies as the number of neighbors is increased.

Figure 4.2 shows that the accuracy increases substantially as the number of neighbors increases. This shows that the classifier predicts better if it looks at multiple consecutive alignment columns instead of a single alignment column. We can also observe that for the overall test accuracy, the rate of increase is gradually decreasing as the number of neighbors increase and so we can expect the accuracy to not increase drastically with further increase in number of neighbors. The amount of computer memory required to run these experiments increases with the number of neighbors. With approximately 2.8 million training examples, it became impractical to run experiments with more than 80 neighboring columns on this feature set, because the amount of RAM needed was more than 64 GB.

The best overall test accuracy obtained by Random Forest for feature set 0 is 56.8% with 80 neighbors.

4.1.2 Feature Set 1

For feature set 1, Table 4.3 and Figure 4.3 show the change in test accuracies with increase in number of neighbors.

4.1 Random Forest Results

No. of Neighbors	Overall Accuracy	Class 0 (CM) Accuracy	Class 1 (CG) Accuracy	Class 2 (OA) Accuracy	Class 3 (UA) Accuracy	Class 4 (MA) Accuracy
0	51.17%	53.20%	55.51%	43.23%	60.18%	43.70%
20	53.62%	60.26%	57.57%	47.34%	62.41%	40.51%
40	54.90%	60.27%	58.13%	50.10%	64.15%	41.76%
60	56.16%	60.91%	58.14%	51.08%	67.04%	43.52%
80	57.08%	60.53%	58.08%	53.38%	68.48%	44.78%

Table 4.3: Random forest results: feature set 1. All the accuracies represented here are test set accuracies.

Figure 4.3: No. of neighbors vs test accuracy for data in Table 4.3. The plot shows the change in accuracies as the number of neighbors is increased.

As observed for feature set 0 accuracies increase with increase in the number of neighbors. Again due to large number of training examples and limited computer memory, we couldn't test it with number of neighbors larger than 80. The best overall accuracy obtained with feature set 1 is 57.08% for 80 neighbors. Feature set 1 performs slightly better than feature set 0. A more detailed comparison will be presented in section 4.1.4.

4.1.3 Feature Set 2

For feature set 2, the inferred ancestral nucleotides were added to the previous feature set. Table 4.4 and Figure 4.4 show the results obtained.

No. of Neighbors	Overall Accuracy	Class 0 (CM) Accuracy	Class 1 (CG) Accuracy	Class 2 (OA) Accuracy	Class 3 (UA) Accuracy	Class 4 (MA) Accuracy
0	51.65%	52.83%	56.83%	44.20%	59.97%	44.39%
20	53.84%	60.59%	58.72%	47.86%	61.85%	40.22%
40	55.17%	60.64%	57.56%	50.46%	65.48%	41.61%

Table 4.4: Random forest results: feature set 2. All the accuracies represented here are test set accuracies.

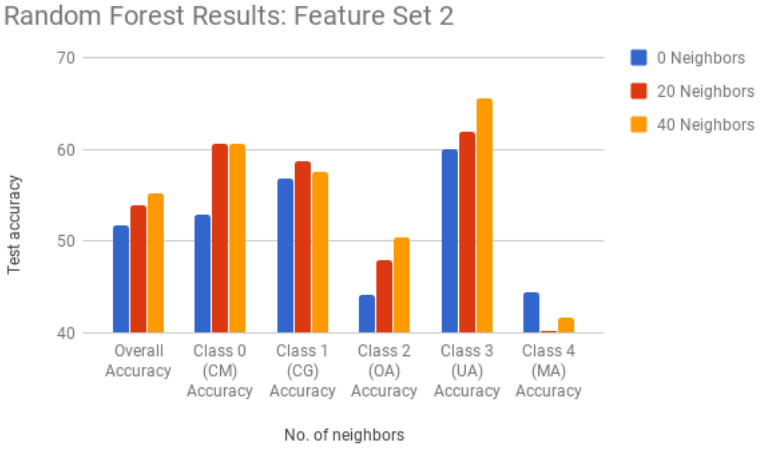


Figure 4.4: No. of neighbors vs test accuracy for data in Table 4.4. The plot shows the change in accuracies as the number of neighbors is increased.

Since we add nine ancestral nucleotides (45 bits after one hot encoding) per alignment column, the training data consumed more memory making it more difficult to increase the number of neighboring columns and at maximum 40 neighbors could be added. We see that the best accuracy achieved by feature set 2 for 40 neighbors is 55.17% which is more than the accuracies of feature set 0 and feature set 1 for 40 neighbors but less than what

was obtained with feature set 1 with 80 neighbors. However, if it was possible to add more number of neighbors in feature set 2, it would perform the best among the feature sets tested so far.

4.1.4 Feature Set 3

The final feature set 3 adds the inferred mutations along the branches of the phylogenetic tree. There are 18 branches in the phylogenetic tree (Figure 3.1) and we are encoding this feature using six bits per branch (Section 3.4.4). Thus in total this feature set will add 108 bits per alignment column. Due to the large number of features per alignment column, it became difficult to add a larger number of neighboring columns. With the entire training set, we could add only 10 neighboring columns. To compare the results with other feature sets for 20 and 40 neighboring columns the number of examples in the training data were reduced to half and one-fourth respectively. Table 4.5 and Figure 4.5 show the results.

No. of Neighbors	**Overall Accuracy**	**Class 0 (CM) Accuracy**	**Class 1 (CG) Accuracy**	**Class 2 (OA) Accuracy**	**Class 3 (UA) Accuracy**	**Class 4 (MA) Accuracy**
0	51.90%	53.64%	57.36%	45.08%	59.80%	43.62%
10	53.51%	60.22%	57.34%	49.49%	61.01%	39.50%
20*	53.27%	59.75%	55.50%	49.60%	62.63%	38.82%
40**	52.74%	59.87%	52.45%	46.71%	65.25%	39.26%

Table 4.5: Random forest results: feature set 3. All the accuracies represented here are test set accuracies. (*) the size of training set was halved. (**) the size of the training set was quartered.

We can see (Figure 4.5) that reducing the size of data set resulted in a decrease in accuracies.

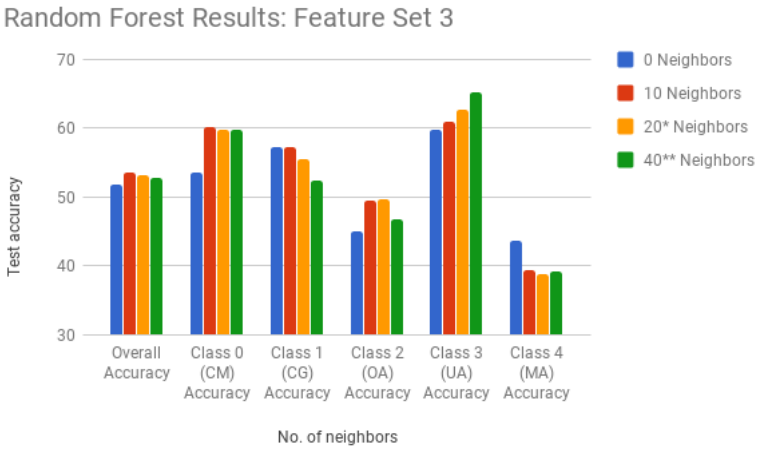


Figure 4.5: No. of neighbors vs test accuracy for data in Table 4.5. The plot shows the change in accuracies as the number of neighbors is increased. (*) the size of the training set was halved. (**) the size of the training set was quartered.

4.1.5 Summary of random forest prediction accuracies

A comparison between all the feature sets was drawn (Figure 4.6) and it was observed that for zero neighbors feature set 3 performed best but for 20 and 40 neighbors feature set 2 performed best. Feature set 1 performed best with more than 40 neighbors. It would not be fair to compare feature set 3 with other feature sets for 20 or more neighbors because the classifier was trained on less data for feature set 3.

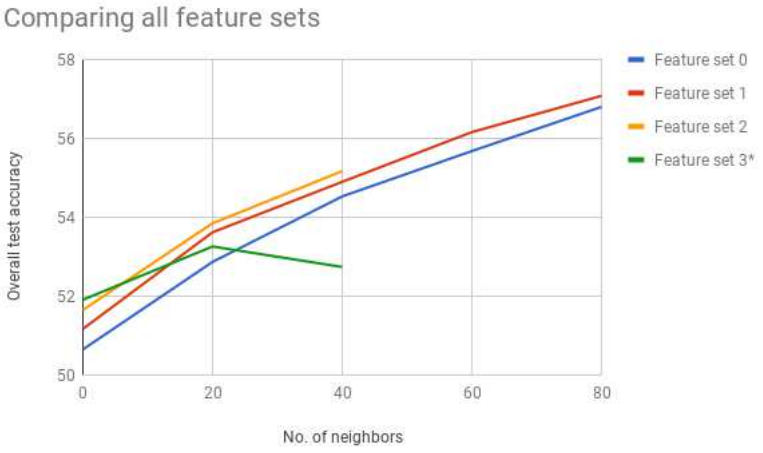


Figure 4.6: Random forest results: comparing all feature sets. The results for feature set 3 are on smaller number of training examples for 20 and 40 neighboring columns.

In conclusion, the experiments with RF showed that all the feature sets engineered for this problem helped the classifier. Feature set 3 gave the best results when the classifier was trained with the entire dataset (zero neighbors). However there is a trade-off between increasing the number of features per alignment column and increasing the number of neighboring columns. It looks like if it was possible to run RF on feature set 3 without compromising on the training data, the best accuracy would be achieved by feature set 3 with 80 neighbors. If we only look at the overall best test accuracy achieved by RF, then feature set 1 with 80 neighboring columns gave the best accuracy of 57.08%. Table 4.6 shows its corresponding confusion matrix.

	Class 0 **(CM)**	**Class** 1 **(CG)**	**Class** 2 **(OA)**	**Class** 3 **(UA)**	**Class** 4 **(MA)**	**Accuracy**
Class 0 **(CM)**	145192	164	37234	40	57234	60.53%
Class 1 **(CG)**	1427	136914	2410	93385	1581	58.08%
Class 2 **(OA)**	60643	1323	123611	415	45570	53.38%
Class 3 **(UA)**	1410	70604	1151	162608	1680	68.48%
Class 4 **(MA)**	90990	729	38898	313	106183	44.78%

Table 4.6: Confusion matrix for best overall test accuracy achieved by RF. Rows represent the true class while columns represent the predicted class. For example, 164 examples of true class 0 are predicted to belong to class 1.

The confusion matrix shows that there is larger confusion among classes 0 (CM), 2 (OA) and 4 (MA), and among classes 1 (CG) and 3 (UA). This is because in classes 0, 2 and 4 a nucleotide is aligned to a nucleotide, while in classes 1 and 3 a nucleotide is aligned to a gap (Refer Table 3.1). Therefore its easier to differentiate between these two group of classes. A completely random predictor would predict each classes equally, but a slightly more clever random classifier would predict classes 0, 2 and 4 with 33% accuracy and classes 1 and 3 with 50% accuracy. The confusion matrix shows that our classifier is significantly better than this and this behavior is observed in all the confusion matrices.

Unfortunately, RF cannot be easily extended to train in batches and it requires the entire dataset to be loaded into computer's memory. This made it difficult to run experiments with the largest feature set. Artificial neural network overcomes this problem and potentially allows us to learn more complex hypotheses. The next section shows its results.

4.2 Artificial Neural Networks Results

In section 2.2.2, we saw that neural networks could be trained in batches and this helps us overcome the problem of storing the entire dataset for training (like in random forests). We trained the neural network (with the architecture shown in section 3.6.1) in mini batches of 250 training examples. Each experiment was run for ten epochs of the entire training set. It was observed that the classifier achieved the maximum test accuracy within ten epochs and further training overfit the model. The learning rate was set to 0.001. The loss function

used was sparse categorical cross entropy with adam optimizer. Training approximately 2.8 million examples in batches of 250 for ten epochs solves the memory problem but it results in larger running time for each experiment. The running time of each experiment varied between 12 hours to 96 hours with an average of around 24 hours per experiment.

Random forests showed that feature set 3 performs best without reducing the size of the data-set. For ANN, we experiment with feature sets 0, 1 and 2, matching the number of neighbors in RF to draw a comparison. For feature set 3, we experiment with larger number of neighbors than RF.

4.2.1 Feature Set 0

Table 4.7 shows the results for feature set 0 with the corresponding graph in Figure 4.7. As observed with RF, the overall test accuracy increases with increase in number of neighbors. However, unlike the results observed with RF, we can see in the graph that the overall test accuracy almost plateaus starting at 20 neighbors.

No. of Neighbors	**Overall Accuracy**	**Class 0 (CM) Accuracy**	**Class 1 (CG) Accuracy**	**Class 2 (OA) Accuracy**	**Class 3 (UA) Accuracy**	**Class 4 (MA) Accuracy**
0	51.04%	52.67%	56.40%	42.72%	58.49%	44.90%
20	55.73%	56.47%	60.02%	49.22%	60.61%	52.35%
40	56.00%	54.40%	64.29%	55.48%	58.66%	47.22%
60	55.93%	54.13%	66.64%	53.29%	57.34%	48.29%
80	56.25%	57.30%	63.44%	53.94%	61.65%	44.91%

Table 4.7: Artificial neural network results: feature set 0. All the accuracies presented here are test set accuracies.

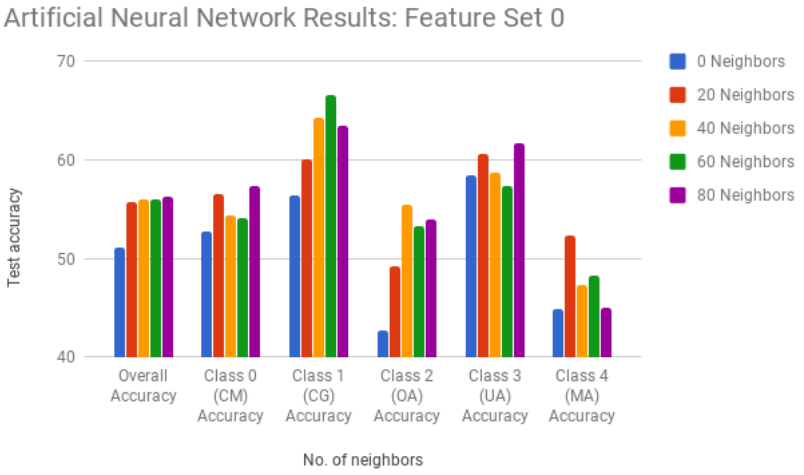


Figure 4.7: No. of neighbors vs test accuracy for data in Table 4.7. The plot shows the change in accuracies as the number of neighbors is increased.

4.2.2 Feature Set 1

The results for feature set 1 is shown in Table 4.8 with its corresponding graph in Figure 4.8. With increase in neighbors, the accuracies for class 0 (CM) and class 2 (OA) decrease but it increases in other classes. The overall test accuracy has stopped increasing with increase in neighbors and the best accuracy achieved is 58.8%.

No. of Neighbors	**Overall Accuracy**	**Class 0 (CM) Accuracy**	**Class 1 (CG) Accuracy**	**Class 2 (OA) Accuracy**	**Class 3 (UA) Accuracy**	**Class 4 (MA) Accuracy**
0	50.94%	50.38%	58.16%	43.10%	56.99%	46.09%
20	57.49%	63.93%	63.11%	58.02%	59.27%	43.12%
40	58.82%	63.22%	63.52%	59.03%	61.02%	47.32%
60	58.85%	62.86%	66.78%	57.96%	58.54%	48.11%
80	58.81%	61.47%	61.80%	56.70%	63.99%	50.09%

Table 4.8: Artificial neural network results: feature set 1. All the accuracies presented here are test set accuracies.

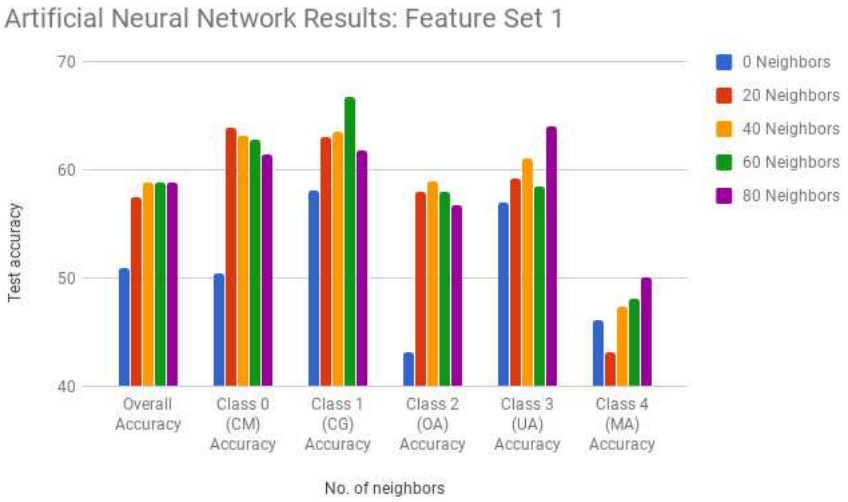


Figure 4.8: No. of neighbors vs test accuracy for data in Table 4.8. The plot shows the change in accuracies as the number of neighbors is increased.

Feature set 1 performs substantially better than feature set 0. The best overall test accuracy is around 2.5% higher for feature set 1 than feature set 0. The comparison will be seen graphically later when we compare all the feature sets (Figure 4.11).

4.2.3 Feature Set 2

Adding inferred ancestral data increased the accuracy considerably in random forests and the same is observed in ANN. The accuracies are shown in Table 4.9 and Figure 4.9. As expected the accuracies increase with increase in number of neighbors.

No. of Neighbors	Overall Accuracy	Class 0 (CM) Accuracy	Class 1 (CG) Accuracy	Class 2 (OA) Accuracy	Class 3 (UA) Accuracy	Class 4 (MA) Accuracy
0	51.98%	53.30%	58.57%	43.00%	58.33%	46.73%
20	60.44%	65.45%	65.40%	62.61%	59.69%	49.07%
40	62.70%	70.90%	62.57%	64.11%	65.42%	50.48%
60	64.02%	72.74%	60.72%	69.67%	69.59%	47.35%

Table 4.9: Artificial neural network results: feature set 2. All the accuracies presented here are test set accuracies.

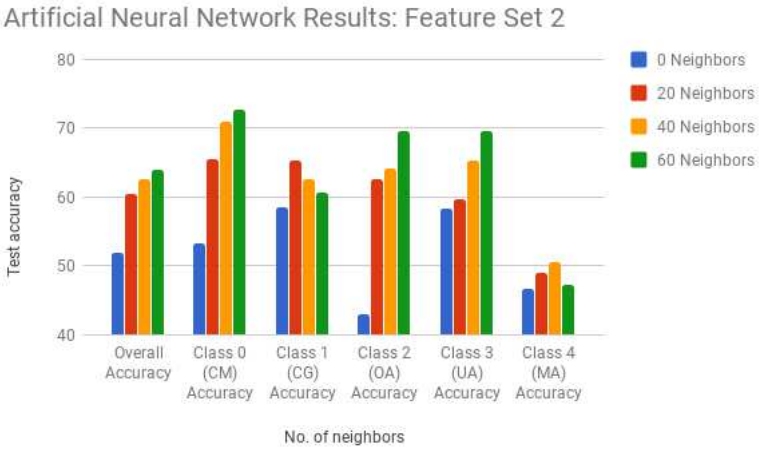


Figure 4.9: No. of neighbors vs test accuracy for data in Table 4.9. The plot shows the change in accuracies as the number of neighbors is increased.

Feature set 2 performs substantially better than feature set 1. There is an increase of around 6% between the best overall test accuracy for feature set 2 than feature set 1. This shows that the inferred ancestral nucleotides helped the classifier considerably.

4.2.4 Feature Set 3

For feature set 3 first we experiment with the same numbers of neighbors as with RF to compare the two algorithms, and then we increase the number of neighbors further than

that, to obtain the most accurate model. Table 4.10 shows the results with the corresponding graph in Figure 4.10.

No. of Neighbors	Overall Accuracy	Class 0 (CM) Accuracy	Class 1 (CG) Accuracy	Class 2 (OA) Accuracy	Class 3 (UA) Accuracy	Class 4 (MA) Accuracy
0	52.33%	53.60%	60.28%	44.66%	56.99%	46.12%
20	61.19%	72.60%	66.33%	62.96%	58.36%	45.68%
40	63.04%	73.74%	59.36%	68.00%	68.87%	45.25%
60	64.15%	76.33%	59.03%	69.90%	71.23%	44.27%
120	65.04%	78.01%	58.88%	67.39%	73.16%	47.76%

Table 4.10: Artificial neural network results: feature set 3. All the accuracies presented here are test set accuracies.

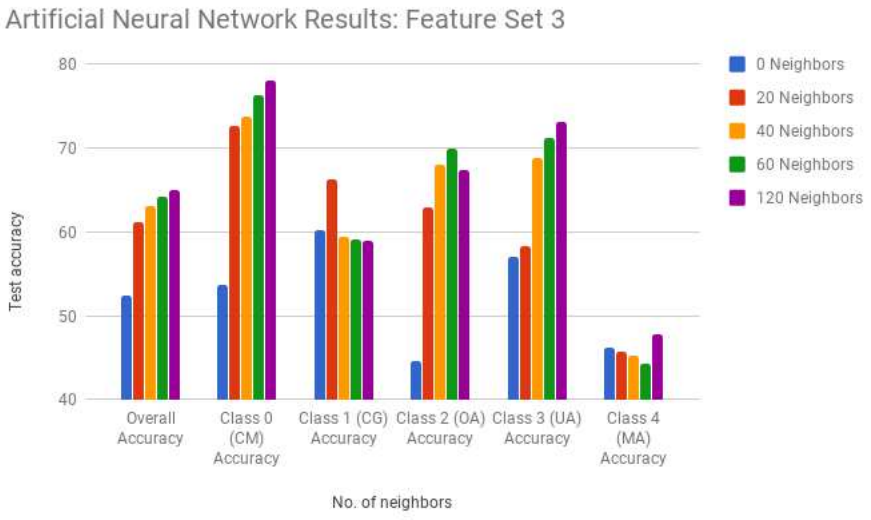


Figure 4.10: No. of neighbors vs test accuracy for data in Table 4.10. The plot shows the change in accuracies as the number of neighbors is increased.

The increase in overall accuracy from 60 to 120 neighbors (i.e. over an increase of 60 neighbors) is small compared to the increase from 40 to 60 neighbors (i.e. over an increase of 20 neighbors), thereby indicating that adding more neighbors is unlikely to

greatly increase the accuracy. Also the running time required to run an experiment with feature set 3 was the highest among all feature sets because of the number of features per alignment column. Therefore, we did not experiment with more neighbors. Among the four feature sets, feature set 3 performed the best (Figure 4.11) even though the difference in accuracies between feature set 3 and feature set 2 is small.

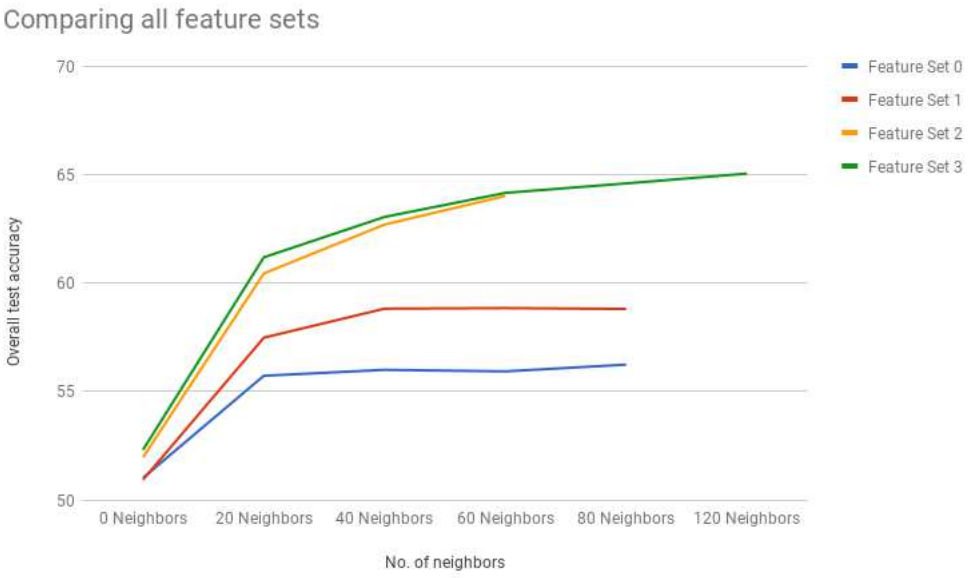


Figure 4.11: Artificial neural network results: comparing all feature sets.

The best accuracy obtained using ANN is 65.04% and its corresponding confusion matrix is displayed in below Table 4.11.

	Class 0 **(CM)**	**Class** 1 **(CG)**	**Class** 2 **(OA)**	**Class** 3 **(UA)**	**Class** 4 **(MA)**	**Accuracy**
Class 0 **(CM)**	185463	0	19568	0	32711	78.01%
Class 1 **(CG)**	22	136807	39	95455	20	58.88%
Class 2 **(OA)**	33511	7	153956	1	40975	67.39%
Class 3 **(UA)**	33	63109	15	172233	41	73.16%
Class 4 **(MA)**	90180	6	32444	8	112146	47.76%

Table 4.11: Confusion matrix for best overall accuracy achieved by ANN. Rows represent the true class while columns represent the predicted class.

As seen in RF, the confusion matrix shows that the classifier is confused primarily between two group of classes, classes 0 (CM), 2 (OA) and 4 (MA) and classes 1 (CG) and 3 (UA). We know classes 0 (CM) and 1 (CG) indicate correctly aligned columns while classes 2 (OA), 3 (UA) and 4 (MA) indicate errors in alignment columns. Its important that our classifier makes fewer errors for classes 2 (OA), 3 (UA) and 4 (MA) than classes 0 (CM) and 2 (CG) because predicting a correctly aligned column as incorrect is less severe than missing to detect an incorrectly aligned column. If we consider it as a binary classification problem with classes 0 (CM) and 1 (CG) as positive classes while classes 2 (OA), 3 (UA) and 4 (MA) as negative classes then the multi class confusion matrix can be transformed into a binary confusion matrix as seen in Table 4.12.

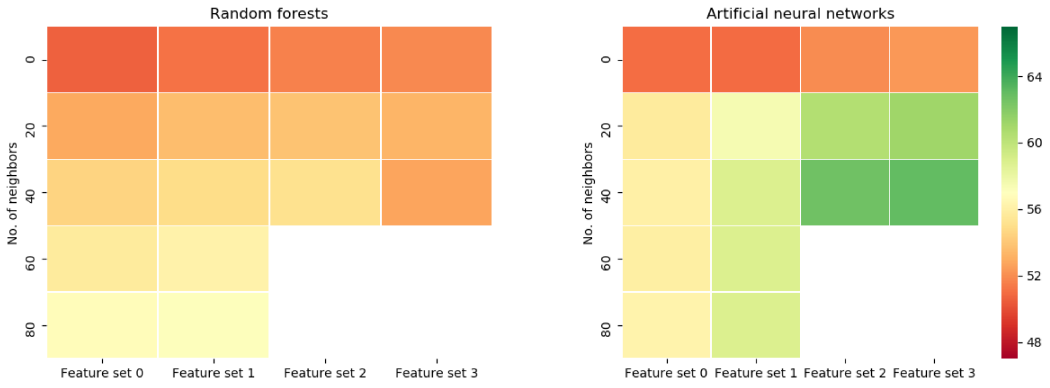
	Positive Class	**Negative Class**
Positive Class	322292	147793
Negative Class	186846	511819

Table 4.12: Binary confusion matrix for the multi class confusion matrix. Classes 0 (CM), 1 (CG) are annotated as positive while other classes are annotated as negative. Rows represent the true class while columns represent the predicted class.

Based on the binary confusion matrix, the false positive rate of the predictor is 0.27, which is less than the false negative rate of 0.31. Thus it predicts more examples of positive class to negative than vice versa, as desired. To conclude, ANN gives a decent accuracy of 65%. The next section compares the results obtained with RF to those obtained with ANN.

4.3 Random Forests vs Artificial Neural Network

We compare the two classifiers based on their overall test accuracies obtained for each feature set with different number of neighboring columns. Figure 4.12 shows this comparison.



(a) Random forests

(b) Artificial neural networks

Figure 4.12: Comparison of RF and ANN accuracy. The heat map plots the overall test accuracy obtained for the different feature sets and number of neighbors. In RF, for feature set 3 the experiments were run with half dataset for 20 neighbors and one-fourth dataset for 40 neighbors.

In ANN, the accuracy for 120 neighbors with feature set 3 is not shown because we do not have the corresponding accuracy in RF. The heat maps show that the accuracy obtained in ANN is almost always higher than RF with a maximum increase of about 10%. We can also observe in each heat map the accuracy almost always increases as we move from feature set 0 to feature set 3. The same observation can be made when we move from 0 to 80 neighbors in each feature set.

In the next section we show the results obtained when the best learned model is used to predict class labels for a real multiple sequence alignment.

4.4 Real Alignment Results

We used the most accurate machine learning model (ANN for feature set 3 with 120 neighbors) to predict the class labels for around a million positions of the 100-way multiple alignment available at the UCSC genome browser [90]. The million base pairs chosen were from human index position 29160498 to 30161782 of chromosome 22.

First, the tool ancestors [86] was used to generate the inferred ancestral nucleotides for

the 100-way alignment file. The alignments of all other species except the ten leaf species and their ancestors (shown in Figure 3.1) were removed. The nucleotides of alignment columns corresponding to the desired human index positions were used to create the necessary features of feature set 3. The features of each alignment column along with 120 neighboring columns were fed into the classifier. The classifier predicted the probability of each class label. The label with the highest probability was chosen as the predicted label for that index position. Table 4.13 shows the statistics for the predictions made by the classifier.

Total no. of alignment columns	**No. of alignment columns predicted for each class**				
	CM	**CG**	**OA**	**UA**	**MA**
998698	274307 (27.45%)	328894 (32.93%)	68850 (6.89%)	236314 (23.66%)	90333 (9.05%)

Table 4.13: Statistics for real alignment results.

Our classifier predicts 60% of the region as correctly aligned (sum of class 0 (CM) and class 1 (CG)) while 40% of the region is detected as suspiciously aligned. This is similar to the original class distribution for human-cow alignment in the simulated data (Refer Table 3.4). Though the percentage of suspiciously aligned regions is high, we can analyze regions in the alignment that are predicted to belong to these classes and find valid explanations for these predictions.

4.4.1 Example of correct match prediction

One of the regions where the classifier detect high probability of correct match is shown in Figure 4.13.

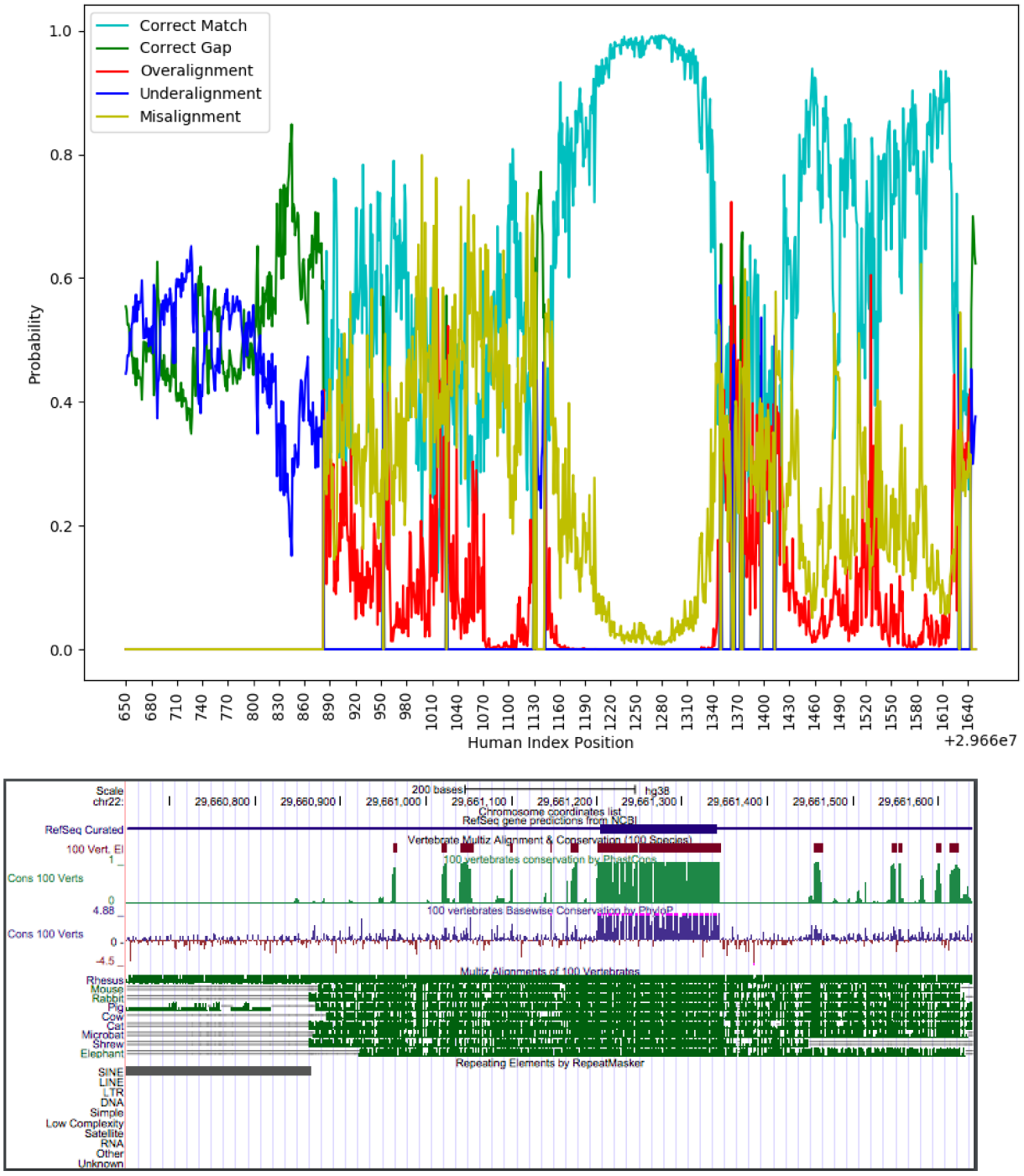
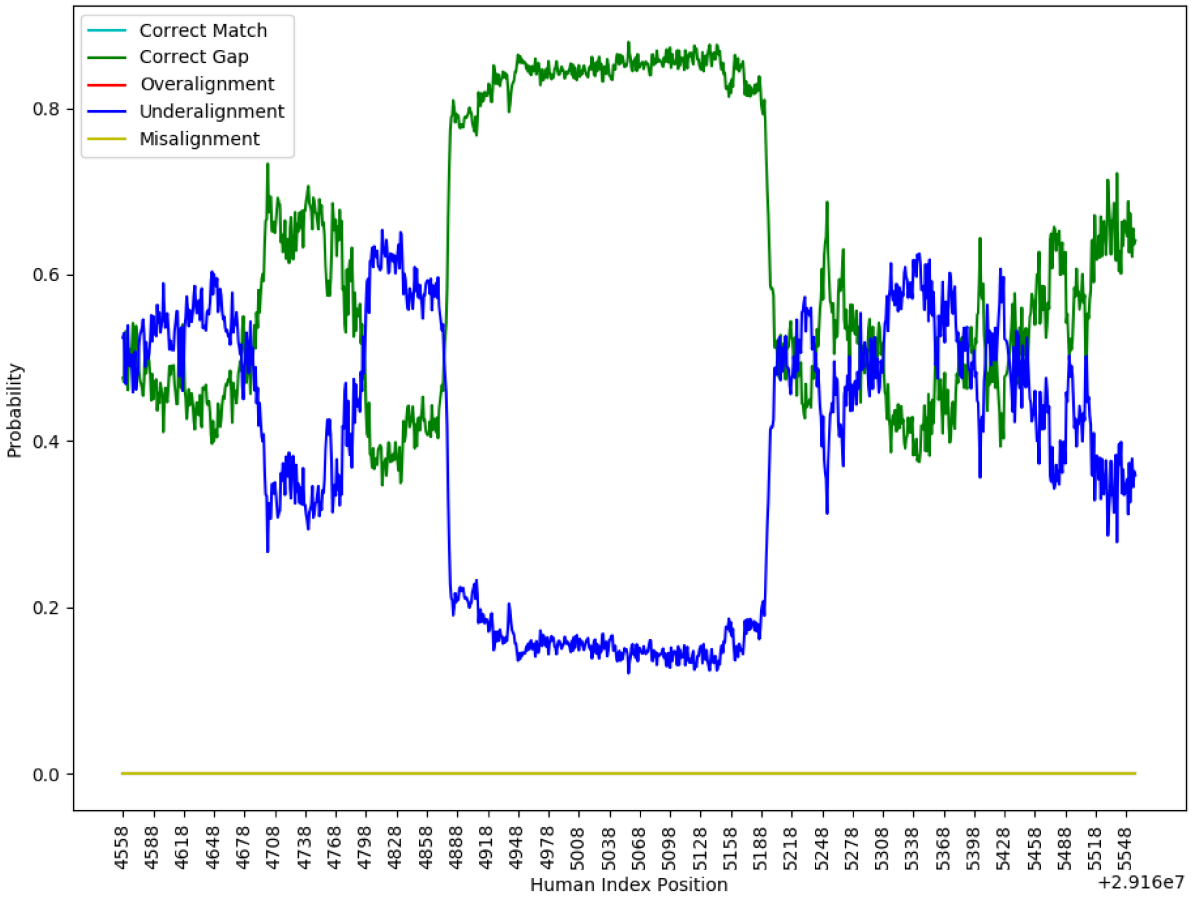


Figure 4.13: Real alignment results: correct match class. The first graph plots the probability of each class label across 1000 index positions. The second figure shows the corresponding region in the UCSC genome browser.

The region where the probability of correct match predicted by our classifier is high corresponds to a highly conserved exonic region in the alignment shown by the genome browser. Highly conserved regions are generally correctly aligned by MSA algorithms and this gives confidence to us that the correct match predictions by our classifier is valid in this particular region.

4.4.2 Example of correct gap prediction

Figure 4.14 shows a region of prediction where the classifier predicts a high probability of correct gap.



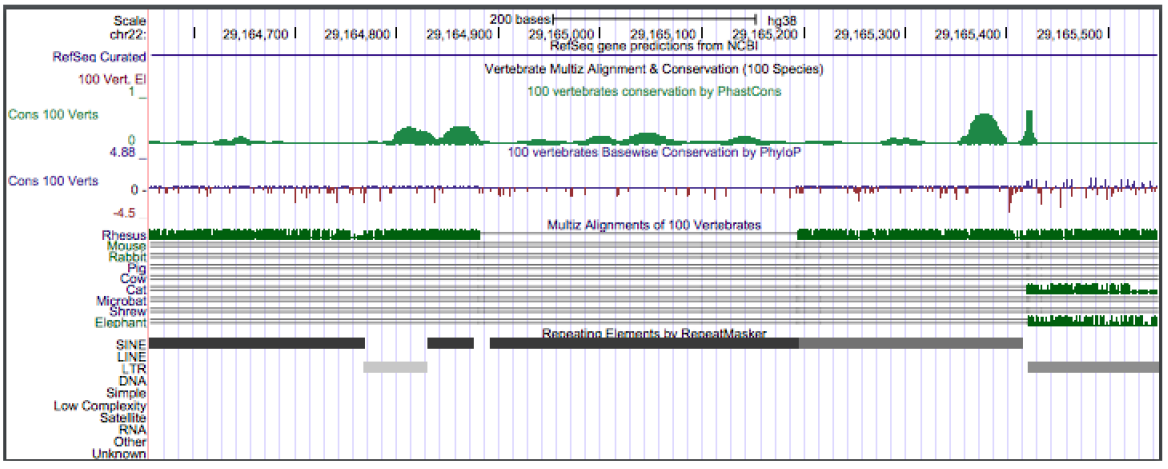
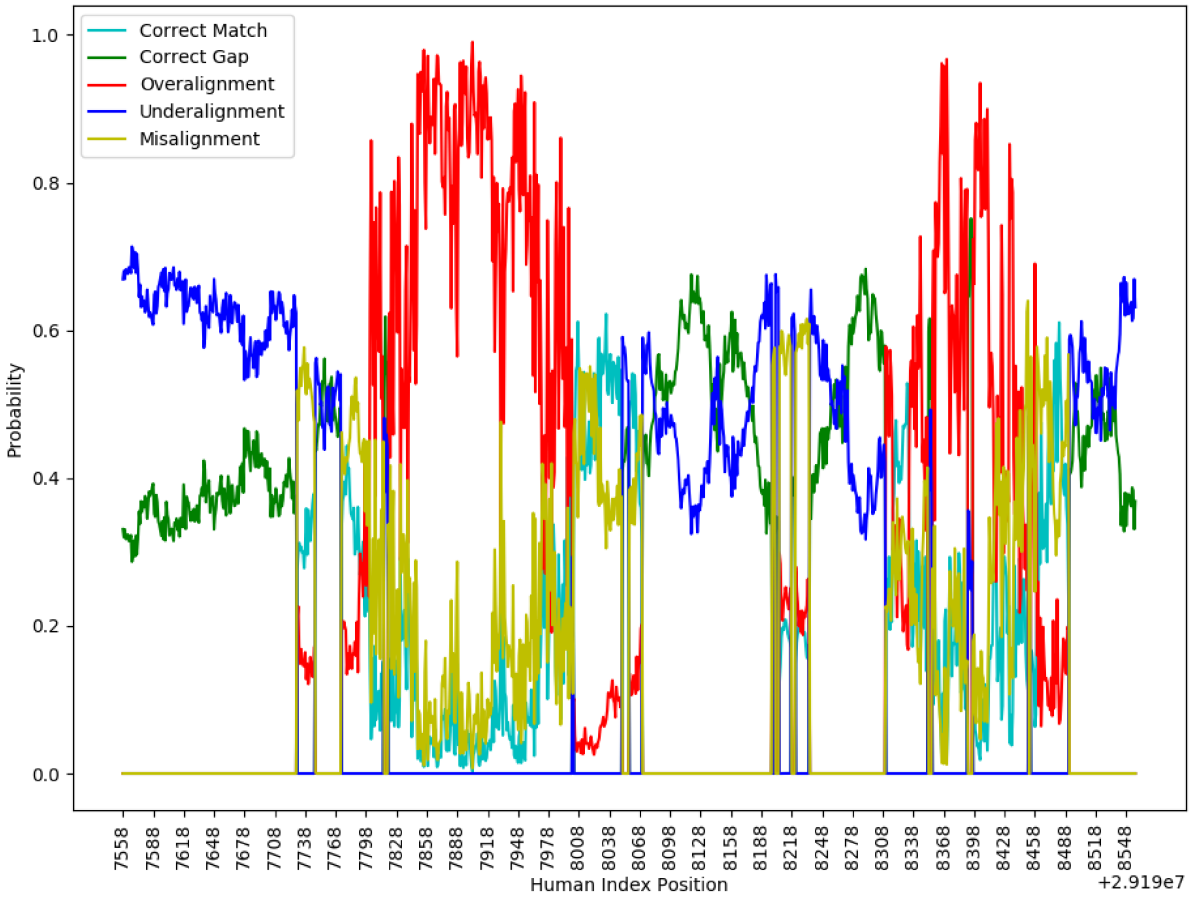


Figure 4.14: Real alignment results: correct gap class. The first graph plots the probability of each class label across 1000 index positions. The second figure shows the corresponding region in the UCSC genome browser.

The region with high probability of correct gap is a region where the nucleotides at the human index position are aligned to gaps in all of the nine other species. Since there are gaps in most of the species, it is more likely that the true alignment is to align the human nucleotides with a gap, which is the prediction made by our classifier. Further increasing our confidence in the correctness of this prediction is the fact that the region where high correct gap predictions are made corresponds to an ALU/SINE transposable element, which is a class of transposable elements that is only active in primates.

4.4.3 Example of over alignment prediction

Figure 4.15 shows a region where the probability of over alignment predicted by our classifier is high.



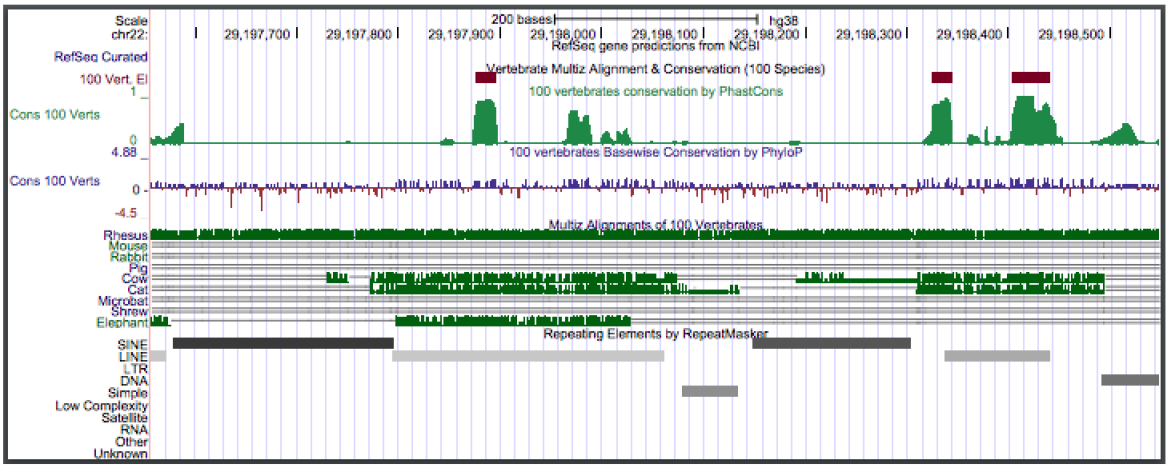


Figure 4.15: Real alignment results: over alignment class. The first graph plots the probability of each class label across 1000 index positions. The second figure shows the corresponding region in the UCSC genome browser.

The region where the probability of over alignment is high corresponds to a region where the human sequence is aligned to gap in most of the species except cow and two other species. The region also contains transposable elements and we know that it is difficult to align transposable elements because they are repeated across the genome. Therefore our classifier is likely correct in predicting that these regions should be aligned to gaps instead of nucleotides.

4.4.4 Example of under alignment prediction

One of the regions where our classifier detects high probability of under alignment is shown in Figure 4.16.

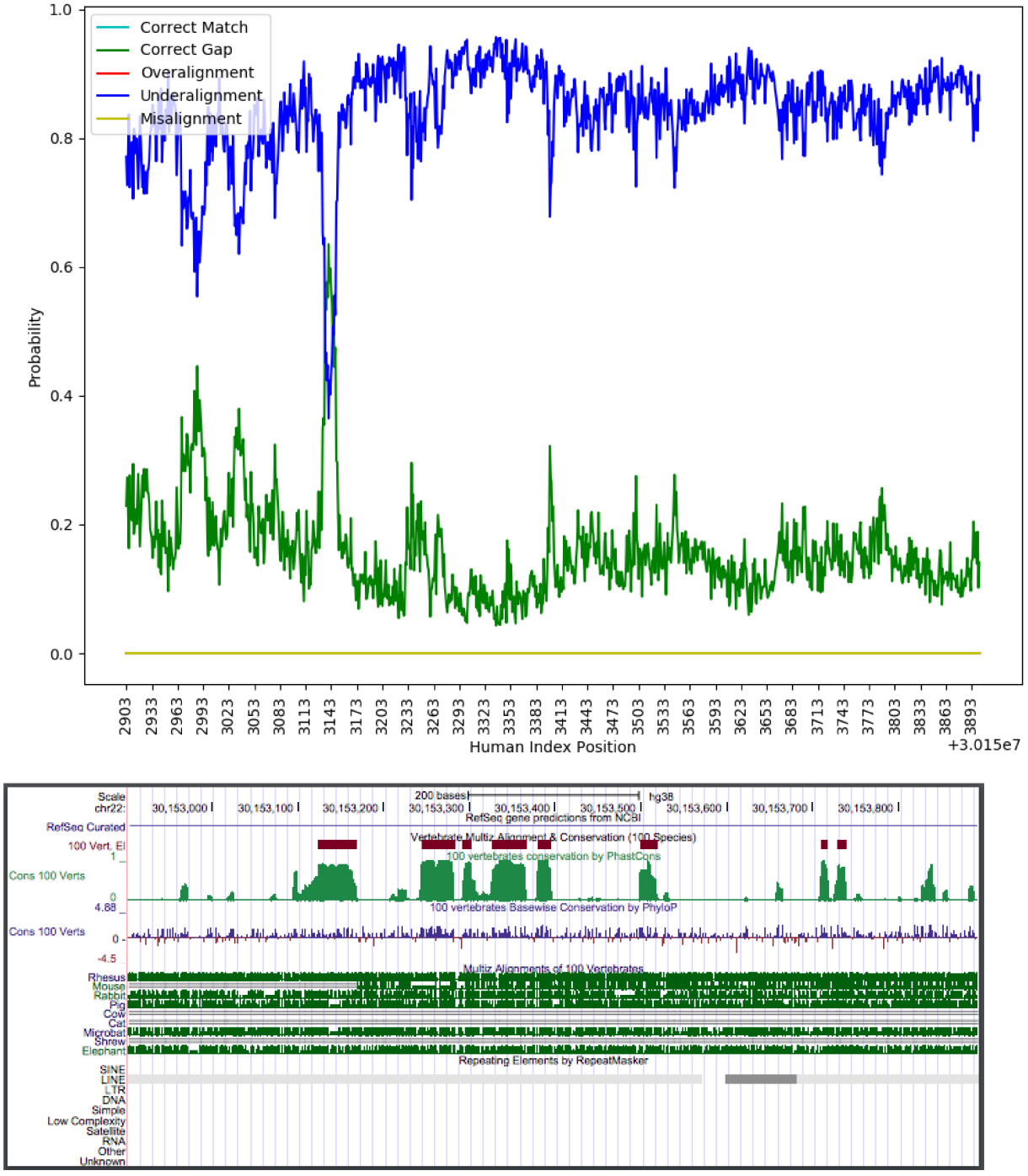


Figure 4.16: Real alignment results: under alignment class. The first graph plots the probability of each class label across 1000 index positions. The second figure shows the corresponding region in the UCSC genome browser.

This is a region that contains several conserved sub regions. This region also contains alignments where the human nucleotides are aligned to nucleotides in many species except cow and a few others. This could be some of the reasons because of which the classifier predicts that the human nucleotides should be aligned to nucleotides in cow rather than gaps.

4.4.5 Example of mis-alignment prediction

Figure 4.17 shows a region of mis-alignment predicted by our classifier.

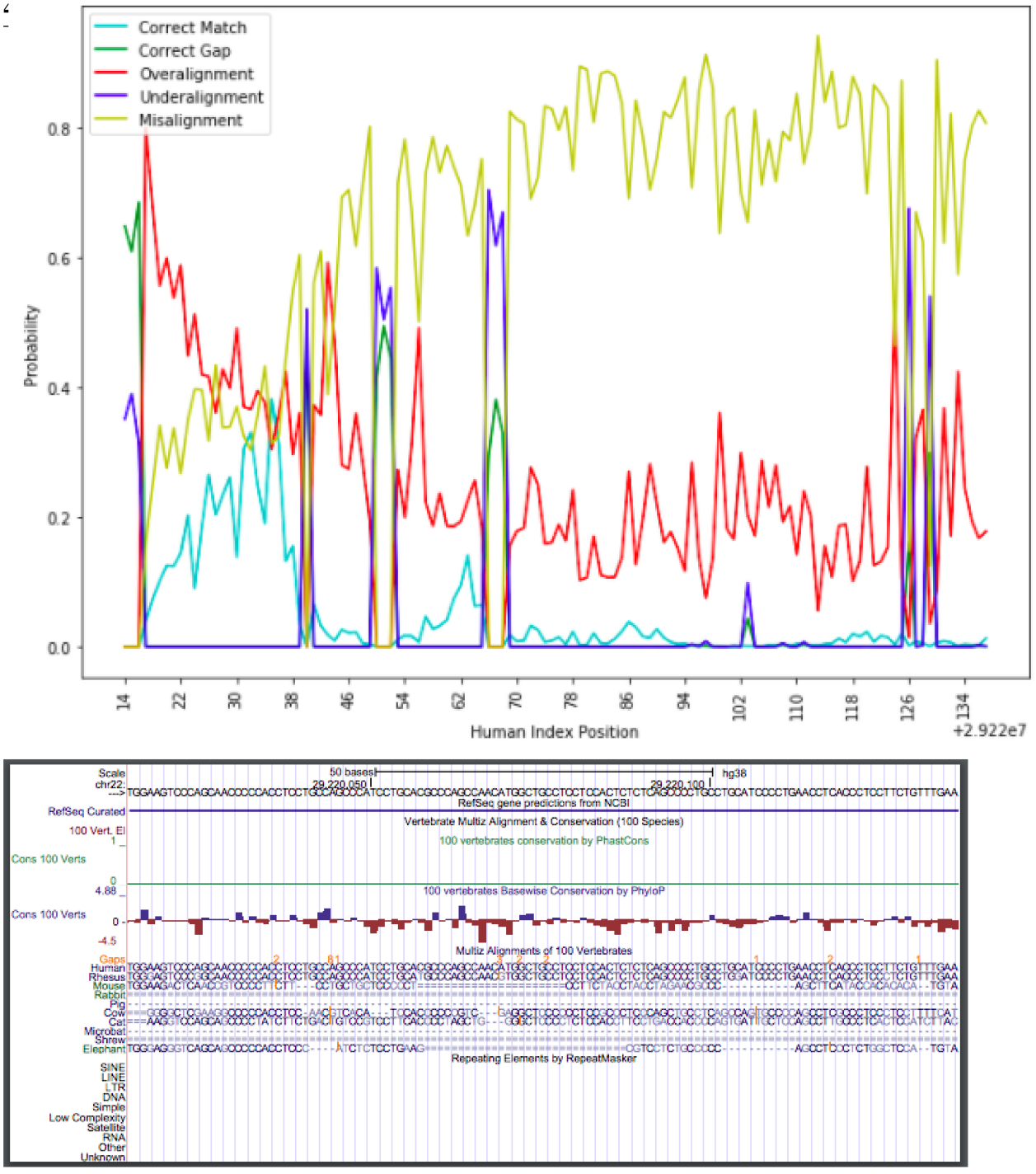


Figure 4.17: Real alignment results: mis-alignment class. The first graph plots the probability of each class label across 122 index positions. The second figure shows the corresponding region in the UCSC genome browser.

The genome browser shows in gray the nucleotides that do not match to the human nucleotides. We can see that many nucleotides in cow do not match to those of human. This could be one of the reasons why our classifier predicts this region to be mis-aligned.

In conclusion, the machine learning model predicts correct and suspicious alignments in a real alignment file, and these predictions could be justified by looking at the annotations and nucleotides at the predicted positions. The next chapter concludes our work with future suggestions.

5

Conclusion and Future Work

Aligning multiple sequences optimally is difficult and the existing algorithms are based on imperfect heuristics. Detecting suspiciously aligned regions in an alignment is critical to researchers who use these alignments for downstream analyses as they can flag these regions or find ways to correct the alignment in these regions. In this thesis, we suggest a machine learning approach that can classify each alignment pair in a MSA to one of the five classes: correct match, correct gap, over alignment, under alignment or mis-alignment. As an example, a machine learning model was built to classify the Human-Cow alignment pair in an alignment of ten vertebrate species. After training the model on simulated data, the model with the highest accuracy (65%) was used to predict the labels on a real alignment. We found around 40% of 1Mbp region to be suspiciously aligned. Some predicted regions were analyzed and it was observed that highly conserved regions were predicted to be correctly aligned and suspicious regions correspond to misaligned regions or regions with transposable elements which makes our predictions more likely to be correct. Our error detection framework is suited for predicting errors only in eukaryotic genomes and cannot be applied to prokaryotic genomes because the phylogenetic tree of prokaryotes does not have a clear tree-like structure due to horizontal gene transfers in their evolutionary process.

In this thesis we used two classifiers: Random Forests (RF) and Artificial Neural Networks (ANN). Some other classifiers such as Support Vector Machines (SVM) could also be used but SVMs do not work well with large datasets as they requires high amount of memory to store the kernel matrix. We chose RF and ANN because RF can serve as a good

baseline model and ANNs are efficient in learning complex hypothesis functions without the need for storing the complete dataset in memory (batch gradient descent).

Although we made significant progress in building a machine learning model to detect errors in an alignment, our method has some limitations that could be improved in the future. Among the fixed set of ten species, our model only predicts the labels for one alignment pair, the human-cow alignment pair because of the class imbalance problem (Section 3.5). We dealt with the class imbalance problem using an under-sampling technique and if we want to predict the class labels for all the alignment pairs using this method we would need to build a classifier for each pair of species. There are other existing methods that can be used to deal with the class imbalance problem, such as penalizing the classifier more if it makes a wrong prediction on an instance of the minority class. The performance of each technique depends on the kind of problem and dataset and there is no way of telling which method is better until they are applied and their results analyzed.

To enhance our feature set we used the inferred ancestral nucleotides but these inferred nucleotides add some uncertainty to our model. For each ancestor in the phylogenetic tree, Ancestors 1.0 [86] predicts the probability of each nucleotide (A, C, G, T or -) at each position. We could use these probabilities in our model to capture this uncertainty. Instead of encoding the ancestor nucleotides in the one-hot encoded form (Table 3.2) we could encode them using the probabilities generated by Ancestors. The mutation along each branch of the phylogenetic tree (Section 3.4.4) could also be encoded by multiplying the appropriate probabilities of the nucleotides at both ends of the branch.

Another limitation of our model is that since we used a simulation-based dataset on a fixed phylogenetic tree, our current model works only for the set of chosen ten species and is not robust enough to predict errors for an alignment with arbitrary number of species. If we want to predict errors for any alignment we would need to train a new model based on the input. However, if required we can predict the errors on an alignment that is used by researchers frequently, such as the 100-way real alignment dataset on the UCSC Genome browser. Although, based on our current implementation simulating the dataset and training a model for this would take months.

Several directions could be fruitful to explore to improve our approach. First, we could

also try to design better architectures for ANN by improving the choice of hyper-parameter values using cross validation. Finding the correct set of hyper-parameters such as the number of hidden layers, the number of neurons per layer, the activation function, the number of epochs, the weight initialization method, etc. can take significant time and computational resources. Second, since the label of one alignment column depends on the neighboring columns, we could also try other advanced neural network techniques, such as Long Short Term Memory (LSTM) networks [91] that are efficient at detecting patterns in sequential data. LSTMs coupled with convolutional neural networks may learn interesting spatial patterns between the alignments columns in an alignment block and may result in more accurate models. Third, since we have the luxury of generating as much training data as we want (caveat: running time and storage space), we can generate data that could be helpful in learning more complex hypotheses using advanced machine learning algorithms (such as LSTM). Finally, after detecting suspiciously aligned regions in whole-genome alignments, we could try to fix these regions by using existing human-computing algorithms such as PHYLO [92, 93]. Ramchalam K.R. (another student in Blanchette's lab) and I are working on a technique that uses reinforcement learning to fix these suspicious alignment blocks.

In conclusion, we propose a unique machine learning approach to address a key problem in bioinformatics. Our approach involves novel use of simulation tools to produce data and after careful feature engineering we apply our model to real alignment data to show that it can be used to detect suspicious alignments. We hope this method will be useful for the research community.

Bibliography

[1] https://www.khanacademy.org/science/biology/her/tree-of-life/a/phylogenetic-trees [Online; accessed 21-May-2018.]

[2] Lander E.S., Linton L.M., Birren B., Nusbaum C., Zody M.C., et al. 2001. Initial sequencing and analysis of the human genome. *Nature* **409:** 860-921.

[3] Waterston R.H., Lindblad-Toh K., Birney E., Rogers J., Abril J.F., et al. 2002. Initial sequencing and comparative analysis of the mouse genome. *Nature* **420:** 520-62.

[4] Blanchette M. 2007. Computation and Analysis of Genomic Multi-Sequence Alignments. *Annu. Rev. Genomics Hum. Genet.* **8:** 193-213.

[5] https://www.codoncode.com/aligner/ [Online; accessed 11-June-2018.]

[6] Gotoh O. 1999. Multiple sequence alignment: Algorithms and applications. *Adv. Biophys.* **36:**159-206.

[7] Wang L, Jiang T. 1994. On the complexity of multiple sequence alignment. *J. Comput. Biol.* **1:** 337-48.

[8] Brudno M., Malde S., Poliakov A., Do C.B., Couronne O., et al. 2003. Glocal alignment:finding rearrangements during alignment. *Bioinformatics* 19 (Suppl. 1): i54-62.

[9] Needleman S. B. and Wunsch C. D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* **48:** 443-453.

[10] Smith T.F. and Waterman M.S. 1981. Identification of common molecular subsequences. *J. Mol. Biol.* **147(1):** 195-197.

[11] Schwartz S., Kent W.J., Smit A., Zhang Z., Baertsch R., et al. 2003. Human-mouse alignments with BLASTZ. *Genome Res.* **13:** 103-7.

[12] Brudno M., Do C.B., Cooper G.M., Kim M.F., Davydov E., et al. 2003. LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res.* **13:** 721-31.

[13] Bray N., Dubchak I., Pachter L. 2003. AVID: a global alignment program. *Genome Res.* **13:** 97-102.

[14] Delcher A.L., Kasif S., Fleischmann R.D., Peterson J., White O., Salzberg S.L. 1999. Alignment of whole genomes. *Nucleic Acids Res.* **27:** 2369-76.

[15] Kent W.J., Baertsch R., Hinrichs A., Miller W., Haussler D. 2003. Evolution's cauldron: duplication, deletion, and rearrangement in the mouse and human genomes. *Proc. Natl. Acad. Sci. USA* **100:** 11484-89.

[16] Pevzner P., Tesler G. 2003. Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. *Genome Res.* **13:** 37-45.

[17] Darling A.C., Mau B., Blattner F.R., Perna N.T. 2004. Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome Res.* **14:** 1394-403.

[18] Dewey C. Pachter L. https://www.biostat.wisc.edu/~cdewey/mercator/

[19] Gupta S. K., Kececioglu J. D., Schäer J. D. 1995. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *Comput. Biol.* **2:** 459-472.

[20] Lermen M. and Reinert K. 2000. The practical use of the A* algorithm for exact multiple sequence alignment. *J. Comput. Biol.* **7:** 655-671.

[21] Lipman D. J., Altschul S. F. and Kececioglu J. D. 1989. A tool for multiple sequence alignment. *Proc. Natl. Acad. Sci.* U.S.A. **86:** 4412-4415.

[22] Reinert K., Lenhof H.P., Mutzel P., Mehlhorn P. and Kececioglu J. 1997. A branch-and-cut algorithm for multiple sequence alignment. 1^{st} *Annual International Conference on Research in Computational Molecular Biology, RECOMB.* 241-249.

[23] Althaus E. and Canzar S. 2008. LASA: A tool for non-heuristic alignment of multiple sequences. *Bioinformatics Research and Development. BIRD 2008. Communications in Computer and Information Science, vol 13. Springer, Berlin, Heidelberg.*

[24] Althaus E., Caprara A., Lenhof H. P., and Reinert K. 2002. Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics. *Bioinformatics 18 Suppl* **2:** S4S16.

[25] Althaus E., Caprara A., Lenhof H.P. and Reinert K. 1999. A polyhedral approach to sequence alignment problems. *PhD thesis, Universität Saarbrücken.*

[26] Durbin R., Eddy S., Krogh A., Mitchison G. 1998. Biological Sequence Analysis:Probabilistic Models of Proteins and Nucleic Acids. *Cambridge Univ. Press.*

[27] Sokal R. R. and Michener C. D. 1958. A statistical method for evaluating systematic relationships. *Univ. Kansas Sci. Bull.* **38:** 1409-1438.

[28] Saitou N. and Nei N. 1987. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.* **4:** 406-425.

[29] Chenna R., Sugawara H., Koike T., Lopez R., Gibson T.J., et al. 2003. Multiple sequence alignment with the Clustal series of programs. *Nucleic Acids Res.* **31:** 3497-3500.

[30] Higgins D.G., Sharp P.M. 1988. CLUSTAL:A package for performing multiple sequence alignment on a microcomputer. *Gene* **73:** 237-44.

[31] Blanchette M., Kent W.J., Riemer C., Elnitski L., Smit A.F., et al. 2004. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Res.* **14:** 708-15.

[32] Miller W., Rosenbloom K., Hardison R.C., Hou M., Taylor J., Raney B., Burhans R., King D.C., Baertsch R., Blankenberg D., et al. 2007. 28-way vertebrate alignment and conservation track in the UCSC Genome Browser.*Genome Res* **17**: 1797-1808.

[33] Meyer L.R., Zweig A.S., Hinrichs A.S., Karolchik D., Kuhn R.M., Wong M., Sloan C.A., Rosenbloom K.R., Roe G., Rhead B., et al. 2013. The UCSC Genome Browser database: extensions and updates 2013. *Nucleic Acids Res* **41:** D64-D69.

[34] Bray N. and Pachter L. 2004. MAVID: Constrained ancestral alignment of multiple sequences. *Genome Res.* **14:** 693-99.

[35] Kent W.J., Sugnet C.W., Furey T.S., Roskin K.M., Pringle T.H., Zahler A.M., Haussler D. 2002. The human genome browser at UCSC. *Genome Res.* **12(6):** 996-1006.

[36] Chakrabarti S., Lanczycki C.J., Panchenko A.R., Przytycka T.M., Thiessen P.A., Bryant S.H. 2006. State of the art: refinement of multiple sequence alignments. *BMC Bioinform.* **7:** 499.

[37] Wang C., Lefkowitz E.J. 2005. Genomic multiple sequence alignments:refinement using a genetic algorithm. *BMC Bioinform.* **6:** 200.

[38] Loytynoja A., Milinkovitch M.C. 2001. SOAP, cleaning multiple alignments from unstable blocks. *Bioinformatics.* **17:** 573-74.

[39] Iantorno S., Gori K., Goldman N., Gil M., Dessimoz C. 2014. Who watches the watchmen? An appraisal of benchmarks for multiple sequence alignment. *Methods Mol. Biol.* **1079:** 59-73.

[40] Stoye J., Evers D., Meyer F. 1997. Generating benchmarks for multiple sequence alignments and phylogenetic reconstructions. *Proc. Int. Conf. Intell Syst. Mol. Biol.* **5:** 303-306.

[41] Cartwright R.A. 2005. DNA assembly with gaps (Dawg): simulating sequence evolution. *Bioinformatics (Suppl. 3)* **21:** iii31-iii38.

[42] Varadarajan A., Bradley R.K., Holmes I. 2008. Tools for simulating evolution of aligned genomic regions with integrated parameter estimation. *Genome Biol.* **9:** R147.

[43] Darling A.C.E., Mau B., Blattner F.R., Perna N.T. 2004. Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome Res.* **14:** 1394-1403.

[44] Darling A.C.E., Mau B., Perna N.T. 2010. ProgressiveMauve: multiple genome alignment with gene gain, loss and rearrangement. *PLoS ONE* **5:** e11147.

[45] Beiko R.G., Charlebois R.L. 2007. A simulation test bed for hypotheses of genome evolution. *Bioinformatics* **23:** 825-831.

[46] Dalquen D.A., Anisimova M., Gonnet G.H., Dessimoz C. 2012. ALF- a simulation framework for genome evolution. *Mol. Biol. Evol.* **29:** 1115-1123.

[47] Edgar R., Asimenos G., Batzoglou S., Sidow A. 2009. EVOLVER. http://www.drive5.com/evolver/

[48] Notredame C., Abergel C., Andrade M.A. 2003. Using multiple alignment methods to assess the quality of genomic data analysis. *Bioinformatics and genomes: current perspectives, Wymondham (UK) Horizon Scientific Press* 30-50.

[49] Chang J.M.M., Di Tommaso P., Notredame C. 2014. TCS: a new multiple sequence alignment reliability measure to estimate alignment accuracy and improve phylogenetic tree reconstruction. *Mol. Biol. Evol.* **31:** 1625-1637.

[50] Landan G., Graur D. 2007. Heads or tails: a simple reliability check formultiple sequence alignments. *Mol. Biol. Evol.* **24:** 1380-1383.

[51] Hall B. 2008. How Well Does the HoT Score Reflect Sequence Alignment Accuracy? *Mol. Biol. Evol.* **25(8):** 1576-80.

[52] Landan G., Graur D. 2008. Local reliability measures from sets of co-optimal multiple sequence alignments. *Pac. Symp. Biocomput.* 15-24.

[53] Penn O., Privman E., Ashkenazy H., Landan G., Graur D., Pupko T. 2010. GUIDANCE: a web server for assessing alignment confidence scores. *Nucleic Acids Res.* **38:** W23-W28.

[54] Felsenstein J. 1985. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution* vol. 39 (pg. 783-791).

[55] Katoh K., Misawa K., Kuma K. and Miyata T. 2002. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Res.* **30:** 3059-3066.

[56] Löytynoja A. Goldman N. 2010. webPRANK: a phylogeny-aware multiple sequence aligner with interactive alignment browser. *BMC Bioinformatics.* **11:** 579.

[57] Wu M., Chatterji S., Eisen J.A. 2012. Accounting For Alignment Uncertainty in Phylogenomics. *PLOS ONE* **7(1):** e30288.

[58] Patrick K., Karen M., Johannes D., Birthe T., Björn M. von R., Johann W. W. and Bernhard M. 2010. Parametric and non-parametric masking of randomness in sequence alignments can be improved and leads to better resolved trees. *Frontiers in Zoology.* **7:** 10.

[59] Prakash A. & Tompa M. 2007. Measuring the accuracy of genome-size multiple alignments. *Genome biology.* **8.** R124. 10.1186gb-2007-8-6-r124.

[60] Prakash A., Tompa M. 2005. Statistics of local multiple alignments. *Bioinformatics* **21:** i344 - i350.

[61] Thompson J.D., Koehl P., Ripp R., Poch O. 2005. BAliBASE 3.0: latest developments of the multiple sequence alignment benchmark. *Proteins.* **61:** 127-36.

[62] Edgar R.C. 2004. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinform.* **5:** 113.

[63] Kemena C., Notredame C. 2009. Upcoming challenges for multiple sequence alignment methods in the high-throughput era. *Bioinformatics.* **27:** 2455-2465.

[64] Margulies E.H., Cooper G.M., Asimenos G., Thomas D.J., Dewey C.N., Siepel A., Birney E., Keefe D., Schwartz A.S., Hou M. et al. 2007. Analyses of deep mammalian sequence alignments and constraint predictions for 1% of the human genome. *Genome Res.* **17:** 760-774.

[65] Paten B., Herrero J., Beal K., Fitzgerald S., Birney E. 2008. Enredo and Pecan: genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome Res.* **18:** 1814-1828.

[66] Earl D., Nguyen N., Hickey G., Harris R.S., Fitzgerald S., et al. 2014. Alignathon: a competitive assessment of whole-genome alignment methods. *Genome Res.* **24(12):** 2077-2089.

[67] Earl D., Bradnam K., St. John J., Darling A., Lin D., Fass J., Yu HO., Buffalo V., Zerbino D.R., Diekhans M., et al. 2011. Assemblathon 1: a competitive assessment of de novo short read assembly methods. *Genome Res.* **21:** 2224-2241.

[68] Kim J., Ma J. 2011. PSAR: measuring multiple sequence alignment reliability by probabilistic sampling. *Nucleic Acids Res.* **39:** 6359-6368.

[69] Samuel A.L. 1959. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development, vol. 3, no. 3, pp. 210-229.*

[70] https://djsaunde.wordpress.com/2017/07/17/the-bias-variance-tradeoff/ [Online; accessed 15-May-2018.]

[71] QUINLAN J.R. 1985. Induction of Decision Trees. *Centre for Advanced Computing Sciences, New South Wales Institute of Technology, Sydney, Australia.*

[72] Mitchell T.M. 1997. Machine Learning. Section 3.4.1 Pg 55-60.

[73] Haykin S. 1998. Neural Networks: A Comprehensive Foundation. *Prentice Hall PTR.*

[74] Johnson R. and Zhang T. 2013. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. *Advances in Neural Information Processing Systems* **26:** 315-323.

[75] Kingma DP and Ba JL. 2015. Adam: a Method for Stochastic Optimization. International Conference on Learning Representations. *Third International Conference for Learning Representations, San Diego, 2015*

[76] Duchi J., Hazan E., & Singer Y. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* **12:**2121âĂŞ2159.

[77] Li H., Doermann D. and Kia O. 2000. Automatic text detection and tracking in digital video. *IEEE Transactions on Image Processing* vol. 9, no. 1, pp. 147-156.

[78] Zhang G.P. 2003. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing.* **50:** 159-175.

[79] Stormo G.D. 2000. DNA binding sites: representation and discovery. *Bioinformatics.* **16:** 16-23.

[80] Khan J., Wei J.S, et al. 2001. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature Medicine.* **7:** 673-679.

[81] Brun Y. and Ernst M. D. 2004. Finding latent code errors via machine learning over program executions *Proceedings. 26th International Conference on Software Engineering.* pp. 480-490. doi: 10.1109/ICSE.2004.1317470

[82] Wang A.H. 2010. Detecting Spam Bots in Online Social Networking Sites: A Machine Learning Approach. *IFIP Annual Conference on Data and Applications Security and Privacy.* volume 6166. 335-342.

[83] Buczak A. L. and Guven E. 2016. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials* vol. 18, no. 2, pp. 1153-1176.

[84] Newick format of the 100 way phylogenetic tree. `http://hgdownload.soe.ucsc.edu/goldenPath/hg38/multiz100way/hg38.100way.commonNames.nh`

[85] Smit A.F.A., Hubley R., Green P. 2010. RepeatMasker Open-3.0. `http://www.repeatmasker.org`

[86] Diallo A.B., Makarenkov V., Blanchette M. 2010. Ancestors 1.0: A web server for ancestral sequence reconstruction. *Bioinformatics, Volume 26, Issue 1.* Pages 130-131.

[87] Abadi M., Agarwal A., Barham P., Brevdo E., Chen Z., Citro C., Corrado G.S., Davis A., et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. *Software available from tensorflow.org.*

[88] Chollet F, et al. 2015. https://github.com/fchollet/keras. *GitHub.*

[89] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M., and Duchesnay E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research.* **12:** 2825-2830.

[90] http://genomewiki.ucsc.edu/index.php/Hg38_100-way_conservation_alignment [Online; accessed 15-June-2018.]

[91] Hochreiter S. and Schmidhuber J. 1997. Long Short-Term Memory. *Neural Comput.* **9:** 1735-1780.

[92] Kawrykow A, Roumanis G, Kam A, Kwak D, Leung C, et al. 2012. Phylo: A Citizen Science Approach for Improving Multiple Sequence Alignment. *PLoS One.* **7(3):** e31362.

[93] Kwak D, Kam A, Becerra D, Zhou Q, Hops A, Zarour R, et al. 2013. Open-Phylo: a customizable crowd-computing platform for multiple sequence alignment. *Genome Biol.* **14(10):** R116.