Classifying DNA using Machine Learning

Alexander Baekey
UCF

Md Mahfuzur Rahaman
UCF

Sarinda Samarasinghe
UCF

Abstract

The human body has enough genetic data to take up 150 trillion gigabytes of information. Most of the time, acquiring a complete set of data is extremely difficult, let alone sifting through all of it. In this field, analysis of trends is key to succeed in properly analyzing genetic data. In this project we compared the accuracy of different models in classifying human DNA into 1 of 7 gene families, and then tested the model's effectiveness on two species it had never seen before. The methods used in this project were Natural Language Processing, k-mer permutation counting, and Recurrent Neural Networks and they were all implemented in Python. When the models were used to classify data from species it had not seen in training, the similarities in accuracy to the human test set corresponded to the genetic similarity of the species to humans.

1 Introduction

Completely understanding the genetic data of the species of planet earth is a monumental task, as the amount of genetic data for just one species is incredibly vast. In addition to this, many species' are not mapped completely, partly due to instability of many organisms when taken out of their natural habitat for examination. The study of bioinfomatics seeks to remedy these issues by creating efficient techniques to analyze large amounts of data, as well as making the most of small amounts of data.

The genetic sequences that we will analyze are all permutations of the 4 **base pairs**: adenine, thymine, guanine, and cytosine. These are commonly represented as A, C, T, and G. These pairs form the 'rungs' on the twisted ladder that is DNA, and their differing orders are what make organisms so different from each other. A **k-mer** refers to a string of length k of base pairs. **Genes** are comprised of these DNA sequences, and together with similar genes form **gene families**, or groups of genes with similar biochemical functions.

In this project we attempt to use machine learning methods to classify genetics sequences of amino acids from 7 genetic families. We aim to examine the generalizability of our classifiers to different species whose genetic data was not used in the genetic process.Our models will be trained from human data, and its performance will be tested with chimpanzee and dog sequences as well. Humans have a very similar genetic makeup to that of chimpanzees, with dogs not too far behind. Due to the relative genetic similarities of the species to humans, we expect our classifiers to perform better on humans, then chimps, then dogs.

2 Problem Statement and Goal

In this project, we have two goals:

- Create a classification model that can accurately assign labels of one of the 7 gene families we have chosen to use to an inputted DNA Sequence.
- Analyze the effectiveness of the model when trained on human data when used to predict the gene families of genetic data from chimpanzees and dogs, species with DNA of varying similarities to humans.

3 Background Knowledge

Any living organism contains one or more cells for its existence. The elements of a cell vary based on the type of the organism. The organisms we will cover in our current study are of similar categories. We will use a small but important part of the cells of these organisms for our analysis.

3.1 Basics of DNA and Gene

Cells are the structural, functional, and biological unit of an organism. The organisms we will deal with in this research are all vertebrates, and the cells of a vertebrate contain chromosomes in it which is a package of DNA. DNAs are the carrier of genetic information and they are a sequence of four nucleotides (A, C, G, T). These nucleotides are the building blocks of DNA. A sequence of nucleotides in DNA that encodes various types of information and works as the basic physical unit of inheritance is called Gene. A set of several similar genes formed by duplication of a single original gene or with similar biochemical functions are referred to as Gene families. For example, G-protein coupled receptors [1] (also known as 7TM receptors) are such a gene family which is involved in many similar diseases.

3.2 Encoding methods for DNA sequences

To work with the nucleotide sequences of DNA data, we need to do pre-processing. In most cases, it is not possible to directly feed the sequence information to an ML model. To solve this issue we have three encoding options with which we can convert the four-letter data to other data types.

3.2.1 Ordinal vector

In ordinal vector representation, we first replace all unknown nucleotides to a common letter 'X'. Then assign different integer or floating point values for each type of nitrogen bases to get the ordinal vector representation of a nucleotide sequence. For example, The sequence 'GGATGCCAAAT' could be [2, 2, 0, 3, 2, 1, 1, 0, 0, 0, 3] and [0.75, 0.75, 0.25, 1.00, 0.75, 0.50, 0.50, 0.25, 0.25, 0.25, 1.00] for integer and floating point ordinal encoding respectively.

3.2.2 One-hot encoding

In this encoding method, after replacing all the unknown nucleotides like the previous encoding, a group of bits is used to represent a nitrogen base where all are '0' except one '1'. Using this encoding, the sequence 'GGATGC' would be [(00100), (00100), (10000), (00010), (00100), (01000)].

3.2.3 K-mer counting

This encoding is used to treat the DNA sequence as a language. In this method, the sequence is broken into K overlapping words. It is required to truncate or pad with dummy letters to make the lengths the same for all the sequences. Then join the words to form sentences and apply any Natural Language Processing model (e.g. Bag of Words). 5-mers for an example sequence 'GGATGCCAAAT' will be GGATG, GATGC, ATGCC, TGCCA, GCCAA, CCAAA, and CAAAT if K-mer encoding is used.

By analyzing all the options of using these encoding vectors, we have decided to use the K-mer counting method for our initial implementation. For some machine learning models, we directly fed the whole sequences to the model which does not require the extra encoding step.

3.3 Machine learning methods to analyze genetic data

Machine learning methods can be applied to genetic data for various purposes. It can be used to detect gene-gene interactions [2]. The learning models can also be applied to classify various diseases or patients [3].

Besides, supervised machine learning methods can predict if a sequence is a gene or not [4]. In our current study, we have used the machine learning models to predict the gene family of several genes. We trained our model with labeled data and tested with genes from the same and phylogenetically nearer organisms. The methods we have used are a) Multinomial Naïve Bayes, b) Linear Support Vector Machine, c) Logistic Regression, d) Random Forest Classifier, e) K-Nearest Neighbor classifier, and f) Recurrent Neural Network with Long-Short Term Memory model.

4 Related Work

Currently in our topic area of sequence analysis, there are multiple approaches to sequence classification. One proposed method was a 2-dimensional Convolutional Neural Network (CNN) approach by Elbayad et al [5]. They used an attention model, which focuses on specific aspects of an input sequentially, which works well with long sequences. It is an improvement on standard encoder-decoder techniques, which is useful for inputs of varying lengths. Their use of a 2D CNN used a combination of the input and target sequences as input. Contrasting with our chosen network of Recurrent Neural Networks (RNN), the CNN has temporal connections established between layers, while the RNN has those within layers. This difference allows CNN to compute multiple activations in parallel, while the RNN is forced into sequentially calculate them.

Another implementation that also uses encoder techniques is the EDA method by Gupta et al. [6] They start by training an autoencoder, a neural network that attempts to learn from reducing the dimensionality of input data. It consists of an encoder that reduces the input sequence to an embedding, and a decoder that tries to rebuild the sequence from that embedding. This autoencoder is combined with an analyzer that makes predictions from the embedding. Together the encoder, decoder, and analyzer form the EDA architecture.

5 Dataset Overview

For our analysis, it was required to collect FASTA files containing the gene sequences and label the sequences based on their family. As our main focus is to apply a machine-learning algorithm to test their effects on DNA sequence data, we skipped the data preparation step and collected a similar dataset from Kaggle [7].

5.1 Gene Families

The dataset we have used in our study is three separate lists of gene sequences for human, chimp, and dog DNA. Each of these lists is a collection of 7 gene families which were labeled with distinct numbers to identify different families. These families are numbered from 0 to 6, and function as the class labels for the classifiers. The 7 gene families that were chosen to be analyzed are given in Table 1. There is twice as much human data as chimpanzees, which in turn has twice the amount as dogs. However, the distribution of the 7 gene families is proportional across all three species as pictured in Figure 1.

Table 1: List of gene families in the dataset

#	Gene families or function groups
1	G-protein coupled receptors
2	Tyrosine kinase
3	Tyrosine phosphatase
4	Synthetase
5	Synthase
6	Ion channel
7	Transcription factor

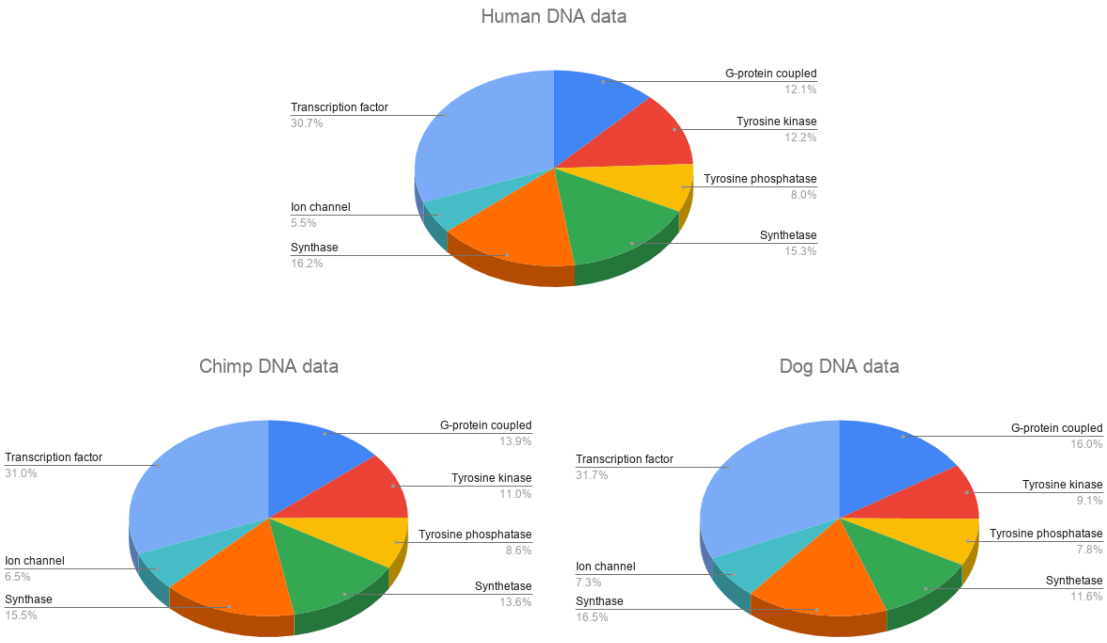


Figure 1: Human, Chimp and Dog DNA data

5.2 Training and Testing Sets

4380 sequences of human DNA were used, 80% of which was used for training, with the rest set aside for testing. For additional testing comparisons, 1682 sequences of chimpanzee DNA and 820 sequences of dog DNA were used. The distribution of training and testing dataset were shown in Figure 2.

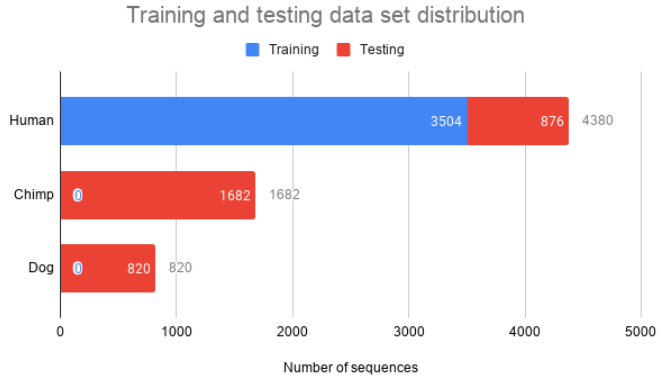


Figure 2: Training and testing data distribution

6 Methodology

We have applied several approaches to analyze our dataset and predict the gene families of the sequences of testing data. Our approaches cover both Supervised Learning and Recurrent Neural Network (RNN). The details of our implementation methods are discussed in the following subsections.

6.1 Supervised Learning

Under supervised learning, we have done the feature extraction part in two different ways. We will refer to them as implementation 1 and implementation 2 respectively.

6.1.1 Implementation 1: Using count vectorizer

In the first implementation, we have used sklearn's Natural Language Processing tools (CountVectorizer) to convert each of the sequences of the training dataset into uniform length feature vectors of 4-gram 6-mer counts. We did the same for both chimp and dog data. Following this approach, we found the feature vectors for each type of data, and the length of these feature vectors was the same. The number of sequences along with their respective feature vector length is shown in Table 2.

Table 2: Obtained feature vector length after applying CountVectorizer

	Sequence count	**Feature vector length**
Human data	4380	232414
Chimp data	1682	232414
Dog data	820	232414

After this step, we have fed these feature vectors to three different machine learning models a) Multinomial Naïve Bayes, b) Linear Support Vector Classifier, and c) Logistic Regression. Multinomial Naïve Bayes algorithm is widely used in Natural Language Processing (NLP) to predict the probability of each word to be the tag of a document. The same strategy works here too. In our case, the features are the counts of each of the tag words (4-gram 6-mers). The Naïve Bayes function uses a parameter named 'alpha' which is an additive (Laplace/Lidstone) smoothing parameter and the value it takes from 0 to 1 (0 for no smoothing). We have tried with separate values for this parameter and found the best result with 'alpha = 0.2'.

Table 3: Confusion matrix for Human data using Multinomial Naïve Bayes

Predicted → Actual ↓	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0	100	0	0	0	1	0	1
1	0	104	0	0	0	0	2
2	0	0	78	0	0	0	0
3	0	0	0	124	1	0	0
4	1	0	0	0	145	0	3
5	0	0	0	0	0	51	0
6	1	0	0	1	0	0	263
Accuracy	98.74%						
Precision	98.75%						
Recall	98.74%						
F1-Score	98.74%						
5-fold cross validation accuracy: 96.32% (+/- 1.38%)							

Linear Support Vector Machine is a robust classifier due to the optimal margin gap between separating hyperplanes. It is more effective in high dimensional spaces and also computationally efficient. We have tried with a maximum of 1000 to 10000 iterations for this algorithm but the results were almost the same. The third model Logistic Regression is an estimator but can be used as a classifier. It works better for larger data set in NLP and also works better when the signal to noise ratio becomes low. We have trained our data using Logistic Regression and tested it with our testing dataset. Logistic Regression implementation in scikit-learn uses a parameter named 'solver' which denotes the algorithm to use in the optimization problem. For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones. For multiclass problems, only 'newton-cg', 'sag', 'saga', and 'lbfgs' handle multinomial loss, 'liblinear' is limited to one-versus-rest schemes. We have tried with all the options and found better results with 'liblinear'. We have also calculated the 5-fold cross-validation to measure the robustness of our training models. The result of our study is discussed in the following section.

6.1.2 Implementation 2: Using k-mer frequencies

In our second implementation, we have used a statistical approach to extract features from the training dataset. We have considered all permutations of 4 nucleotide bases (A, C, G, T). Then we have calculated the frequencies of all these permutations in k-mer (k=4) encoded data set. The 4-mer Permutations without repetition are the following: ACGT, ACTG, AGCT, AGTC, ATCG, ..., TGCA. After getting the frequencies, we have used them as features and fed them to three machine learning models to test the outcomes. The models we have used in this step are a) Logistic Regression, b) Random Forest Classifier, and c) K-Nearest Neighbor Classifier. In this implementation, we found better results with 'newton-cg' as 'solver' algorithm.

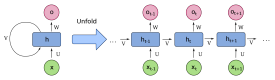
Table 4: Confusion matrix for Human data using Logistic Regression in implementation 2

Predicted → Actual ↓	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0	102	0	0	0	0	0	0
1	0	105	1	0	0	0	0
2	0	0	78	0	0	0	0
3	0	0	0	125	0	0	0
4	0	0	0	0	149	0	0
5	0	0	0	0	0	51	0
6	0	0	0	0	0	0	265
Accuracy	99.89%						
Precision	99.89%						
Recall	99.89%						
F1-Score	99.89%						
5-fold cross validation accuracy: 99.51% (+/- 0.46%)							

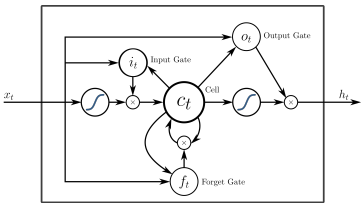
Random forest classifier is mostly suitable for multiclass problems. As our dataset has multiple classes, we took the chance to see the outcome from this classifier. It also works well with a mixture of numerical and categorical features. On the other hand, the KNN classifier works well with small data set and performs better if all of the data have the same scale. In our experiment, we have tried with different values for k in the KNN classifier but got better results using $k = 3$.

6.2 Recurrent Neural Network using Long-Short Term Memory

Recurrent Neural Networks are commonly used in natural language processing (NLP) applications. The structure of a simple unfolded RNN is shown in fig 3a. In NLP, the available training data is often sentences comprised of words from a language. The goal in NLP is either classification such as in sentiment analysis



(a) RNN folds [10]



(b) LSTM cell [11]

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f c_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i c_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o c_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

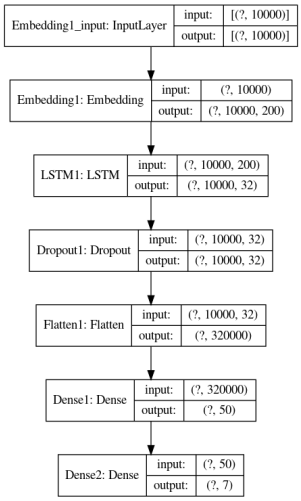
(c) LSTM gate eqns. [11]

Figure 3

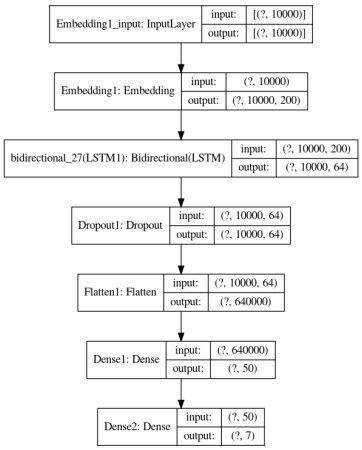
or email spam detection, or it is additional sequence generation, such as with AI chat services. The machine learning technique in NLP works as an interpreter. Classification in NLP is very similar to the task at hand in terms of input information. Treating our sequences as sentences comprised of k-mers words allowed us to process DNA using NLP techniques, namely the recurrent neural network (RNN) with incorporated long short-term architecture (LSTM). [8] This structure can be seen in fig. 3b. Typically in a RNN, within each cell a newly created hidden state vector is combined with a previous state vector to generate output. The hidden state regularly passed along in a RNN network to create the effect of memory. The LSTM works similarly, but with an included "forget" gate. This allow the LSTM to get rid of extraneous information in long sequences. The gate equations are shown in figure 3c. LSTMs unfortunately, still suffer from over-fitting. Overall, because of the dependencies of the components that make up a sequence [9], LSTMs have a high potential for fairly accurate classification of DNA sequences.

The RNN structures for each of the three models are shown in fig. 4. To meet the requirements for training with the Keras Sequential model, the data was truncated to 10,000 words per sequence. Although this meant a significant amount of data wasn't taken into account, this truncation was necessary due to memory constraints. All sequences shorter than 10,000 words were padded with 0's. Following the training requirements, the labels were one-hot encoded. An Embedding layer converts an input tokenized vocabulary of k-mers into a vocabulary of representation vectors. Hexamers were chosen due to their reasonable number of nucleotide combinations. If too small of a vocabulary was chosen, features won't be extracted. If too large of a vocabulary was chosen however, each word would lose its context meaning. Each representation vector is comprised of a unique combination of dynamic weights and corresponds to a single k-mer. As the model is trained, the weights corresponding to each k-mer change based on their occurrence in the input sequences. The human training data as well as the chimp and dog data were all encoded using the same vocabulary set. A 50% Dropout layer was used to mitigate over-fitting. During each update, this layer multiplies a randomized 50% of the entering node values to 0 before moving to the next layer in the network. Given the length of our sequences, the dropout rate had to be set high. [12]

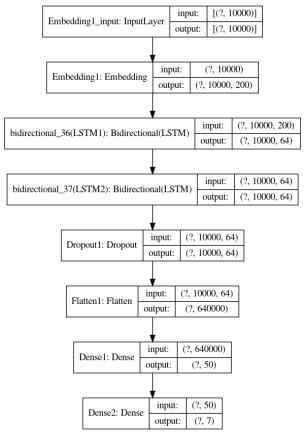
An intermediate dense layer was established between the LSTM structure(s). This was placed for length adaptation. The output of the LSTM (or second LSTM) is a very large flattened vector, so the idea here is to help sort this vector before reaching the output layer. The final dense layer was responsible for classification label output. Because we had 7 labels, one for each of our gene family classes, the final dense layer had 7 nodes. A softmax activation function was placed on this layer to identify the one-hot encoded labels as their original classes. The loss function was set to categorical cross entropy and the optimizer was set to the common 'adam' optimizer. After each epoch the model's callback function was used to compare progress against a checkpoint. If the evaluation accuracy of an epoch was better than the checkpoint, the checkpoint weights were replaced by the ones updated during the most recent epoch.



(a) Vanilla



(b) Bidirect.



(c) Stacked Bidirect.

Figure 4: RNN structures

Stratified Kfolding provided by Sklearn allowed for manual five-fold splitting of training and testing data with a proportional separation of each gene type. Each structure was evaluated under the same hyperparameters, then average prediction accuracy for the chimp and dog sequences were calculated for each model.

Table 5: RNN Results

	Avg Validation Acc	Avg Validation MSE	Avg Chimp Acc	Avg Dog Acc
Vanilla LSTM	86.19%	0.0455	93.33%	74.59%
Bidirectional LSTM	86.53%	0.0461	93.39%	76.07%
Stacked Bidirectional LSTM	86.92%	0.0443	94.21%	76.41%

7 Experimental outcomes

As discussed before, we have tried different parameters for each learning models and keep the results for which we got a better outcome. In the first portion of our implementation we have experimented with Multinomial Naïve Bayes, Linear Support Vector Machine, and Logistic Regression. Out of this three, we got best results with Multinomial Naïve Bayes classifier. The confusion matrix using Naïve Bayes algorithm for human dataset along with the 5-fold validation result is shown in Table 3. We did the same with the features from the second implementation, and we found Logistic Regression performs better for this approach. The confusion matrix using Logistic Regression on implementation 2 for human dataset along with the 5-fold validation result is shown in Table 4. We have generated the confusion matrix for chimp and dog data too, and we have this matrix for not only the best algorithm but also for all experimental methods. We will provide those matrices as supplementary files.

To visually compare the outcomes from our different machine learning algorithms on two different implementations, we have generated a comparative bar chart for both accuracy and precision. We have included the 5-fold cross-validation result in this chart, and we have generated the same chart for both of our implementations of supervised learning. The charts are shown in Figure 5 and 6 for two implementations respectively.

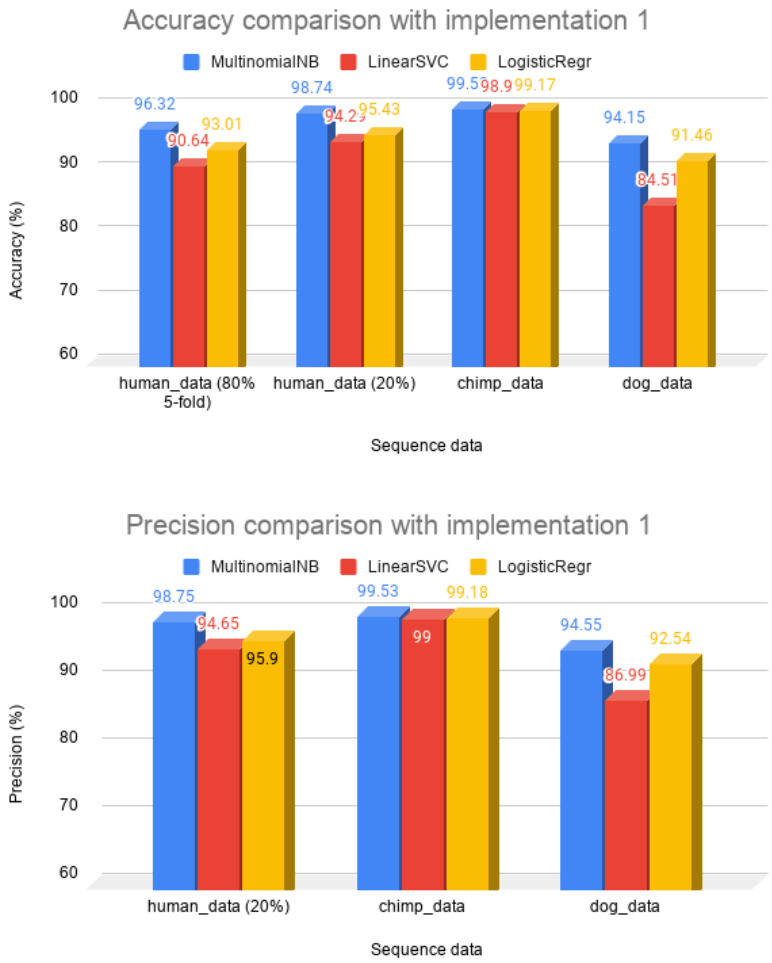


Figure 5: Accuracy and precision comparison with implementation 1

From Figure 5, we can notice the performance of Multinomial Naïve Bayes is quite good for all three testing datasets. It outperforms the Linear SVC and Logistic Regression based on both accuracy and precision. Although the 5-fold cross-validation accuracy is a bit lower than the testing accuracy, 96.32% is high enough to represent the robustness of our learning algorithm for the used dataset. Figure 6 shows the comparison of three classifiers on the features from implementation 2. According to this figure, Logistic Regression outperforms the Random Forest and KNN classifier. Actually, the performance of KNN is very poor compared to the other two for the used dataset. Logistic Regression and Random Forest give more than 99% accuracy and precision. The 5-fold accuracy is also high.

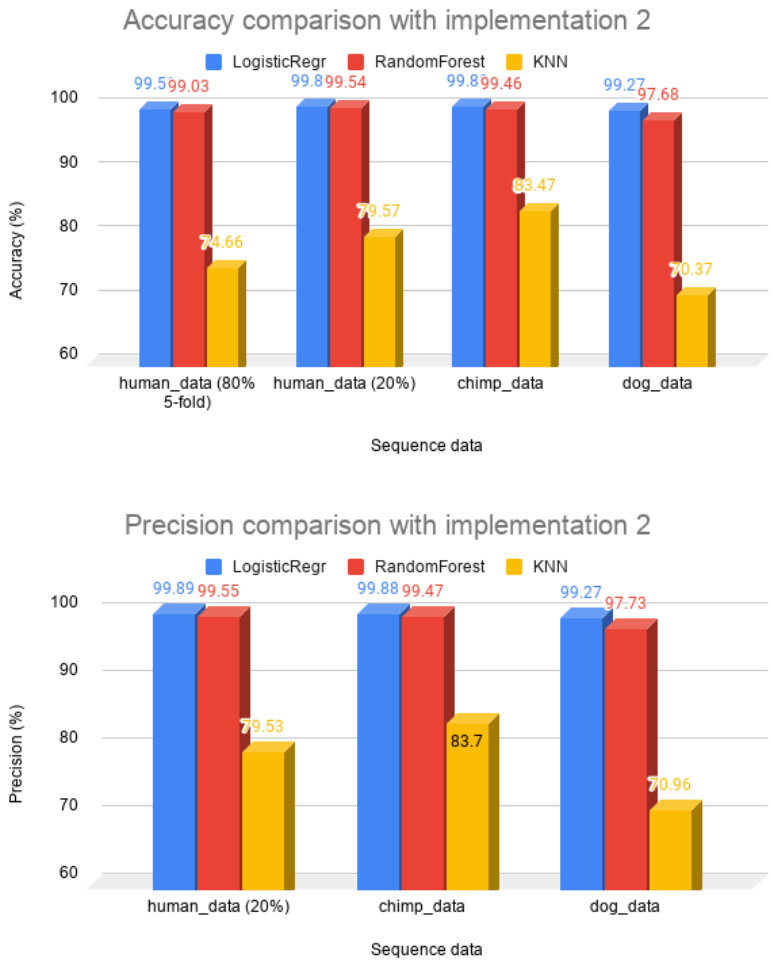


Figure 6: Accuracy and precision comparison with implementation 2

From the accuracy and precision comparison of the two implementations, it is clear that the supervised learning models are working very well for chimp data but, the performance for predicting gene families for dog data is a bit low. It is worth mentioning here that the models were trained with human data and used to predict labels for a part of human, chimp, and dog data. If we take a closer look at the phylogeny tree of humans and the nearest other vertebrates, the reason for the less good performance will be clear. The humans share 98.8% of their DNA sequences with chimpanzees, where the percentage is only 84% for dogs. It seems the DNA sequences of humans and chimp are almost unique, which is the main reason for better performance with chimp data while the training dataset is of human DNA sequences. As dog DNA sequences are less common with humans, the accuracy and precision are comparatively less good for predicting gene families for dog data.

The RNN results were consistent with the literature. Though the validation and prediction accuracy were decent, the model suffered from the length of input data given. If allowed unlimited memory resources, an expanded LSTM with full sequence input would likely have improved performance.

8 Conclusions and Future Work

In this project, we designed multiple classifiers to train on human genetic data, and tested them against DNA sequences from humans, chimpanzees, and dogs. Unsurprisingly, classifications of the chimpanzee data worked better than when using dog data. However, in some instances, when using chimpanzee data for testing, the accuracy was higher than even the human validation data. This can be seen in both method 1 with NLP as well as method 3 with the RNN. This result was very exciting to see as the second goal of this project was to gauge the generalizability of the models to other species.

One large limiting factor in this project was the amount of data available to us. Fortunately, the datasets for chimpanzees and dogs had almost identical distributions of the 7 gene families to the combined training and testing sets for humans. However, only using two additional species in our testing did not allow us to explore our second goal as much as we had hoped. While the genetic similarities between the species did translate well into the relative accuracy over the different testing sets, it would have been better to have 2 or 3 more sets to better analyze the correlation. Unfortunately, over the span of this project, we did not have time for manual data collection from available online genome databases. In a continuation of this project we would be sure to obtain DNA from animals such as rodents and fish, to have a more uniform decrease in genetic similarity to humans over the test species.

References

[1] Daniel M. Rosenbaum, Søren G. F. Rasmussen, and Brian K. Kobilka. The structure and function of g-protein-coupled receptors. *Nature*, 459(7245):356–363, May 2009.

[2] Brett A McKinney, David M Reif, Marylyn D Ritchie, and Jason H Moore. Machine learning for detecting gene-gene interactions. *Applied Bioinformatics*, 5(2):77–88, 2006.

[3] Alberto Romagnoni, , Simon Jégou, Kristel Van Steen, Gilles Wainrib, and Jean-Pierre Hugot. Comparative performances of machine learning methods for classifying crohn disease patients using genome-wide genotyping data. *Scientific Reports*, 9(1), July 2019.

[4] Maxwell W. Libbrecht and William Stafford Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16(6):321–332, May 2015.

[5] Maha Elbayad, Laurent Besacier, and Jakob Verbeek. Pervasive attention: 2d convolutional neural networks for sequence-to-sequence prediction, 2018.

[6] Anvita Gupta and Anshul Kundaje. Targeted optimization of regulatory dna sequences with neural editing architectures. *bioRxiv*, 2019.

[7] Kaggle: Your machine learning and data science community.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[9] Shumin Li, Junjie Chen, and Bin Liu. Protein remote homology detection based on bidirectional long short-term memory. *BMC Bioinformatics*, 18(1), October 2017.

[10] Wikipedia. Recurrent neural network — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Recurrent%20neural%20networkoldid=991832590, 2020. [Online; accessed 06-December-2020].

[11] Wikipedia. Long short-term memory — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Long%20short-term%20memoryoldid=991684445, 2020. [Online; accessed 06-December-2020].

[12] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks, 2016.