DNA Classification with ANNs and HMMs

Katherine Perry

1. Problem and goals

1.1. Broad problem defined at the beginning of the project

In my project, I will investigate the classification and prediction of DNA sequences and structure, as they relate to Hidden Markov Models and Neural Networks . The two strategies for gene prediction are sequence similarity searches and signal-based searches. The current methodology addresses the issue using a Generalized Hidden Markov Model called AUGUSTUS, and techniques such as Dynamic Programming and Neural Networks[6].

HMMs use a transition probability matrix and emission probability matrix as parameters[2]. The inputs are sequences of DNA or other genetic material. The Markov Models can be trained with sequences that are already classified to augment the predictive ability of the parameters[3].

My aim is to compare the existing Hidden Markov Model, Neural Networks for speed, efficacy by training and testing them on two datasets of DNA sequences. I also plan to compare their ability to use BLAST and multiple sequence alignment programs such as CLUSTAL as a tool for integrating extrinsic evidence. To do this I will run BLAST to align the original dataset of sequences, and then attempt to use the alignment to train the model.

Analyzing the accuracy and other metrics of the different machine learning algorithms on the same datasets will show which algorithms are best at nucleotide prediction and DNA classification. I will also try to compare the metrics I find with those of AUGUSTUS.

1.2. Conceptual/technical motivations for choosing this problem

DNA prediction and classification has a vast array of applications in structural genomics, functional genomics, and identification of coding vs non-coding DNA. I am extremely interested in the process and applications, from my biological and computational background. I have worked with Markov Models before, and Hidden Markov Models create another level of complexity and computational power. Artificial Neural Networks similarly have strong learning and predictive power[5].

1.3. Changes in project goals during the semester

Throughout the semester, I had to adjust based on steps that didn't work and results that didn't offer quite the insight I wanted. I decided to focus more on the speed and efficacy of the algorithms than integrating extrinsic evidence. Although I was able to run BLAST on the promoter sequences dataset, the results were not easily integratable into the algorithms. BLAST revealed that the dataset was very similar to the complete genome of Escherichia coli strain

T7Express_LysY chromosome (98.46% identical); however, somehow appending it's genome to the dataset or adding it as a parameter to the ANN algorithm was not feasible.The sequences also had matches in other E. coli and phage target datasets.

2. Datasets

2.1. Data plan at the beginning of the project

I will utilize the UCI Molecular Biology (Promoter Gene Sequences) Data Set as described in a Markov Model, kNN, SVM tutorial I found[1]. The University of California, Irvine has a machine learning repository with ample datasets of DNA sequences. This particular dataset has a list of 56 sequences of nucleotides that have been classified as promoter regions of DNA. A promoter is a region to which proteins bind to start the process of transcription. Each of the sequences are originally formatted as strings but are then manipulated into boolean data. For index i, there are 4 columns i_a, i_c, i_t, and i_g which represent which nucleotide is at that index (indicated by a value of 1). Also, there are class and instance labels for each row that need to be parsed.

Additionally, I will utilize the UCI Molecular Biology Splice-Junction Sequences dataset. The dataset has 3190 sequences of nucleotides that have been classified as splice junctions, which are boundary regions between introns and exons. The dataset is of a similar format as the previous with strings of sequences and also an indication for each sequence whether the region was an EI (exon-intron site) or IE (intron-exon site).

2.2. Progress with data plan

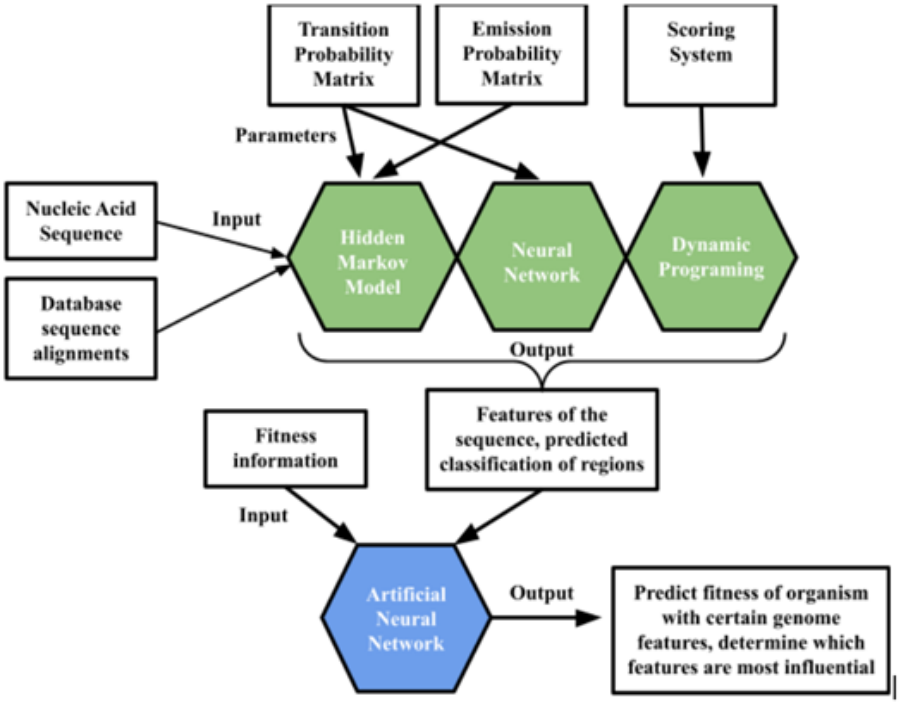
I was able to load both datasets using almost the same process. I utilized pandas to read in the two files via their urls. Then I processed and manipulated the datasets to organize them into usable numeric data with nucleotide values and class values. I had to adjust my data processing approach slightly for the second dataset. The number of rows, columns, and classes were different, so I had to troubleshoot the errors that arose and make small fixes to the code to successfully create a pandas Dataframe object.

2.3. Changes to data plan in the later half of the semester

In accord with my change to the project goals, I also changed my plan to integrate extrinsic evidence to the dataset. I continued using the Promoter and Splice-Junction sequence datasets, but an odd obstacle appeared later in the semester. Because Jupyter Notebook doesn't save the output of the code (the dataframe or accuracy values or variable values) I had to rerun the cells each time I reopened it to work. The second or third time I reran the ANN and other algorithms, the accuracy values of the training and testing plummeted. I'm unsure if the dataset at the url I used to input the sequences changed or if the algorithms simply chose poor training and testing data by chance. Either way, I did record all the accuracy values for the algorithms the first time through for reference in my analysis.

3. Computational approach

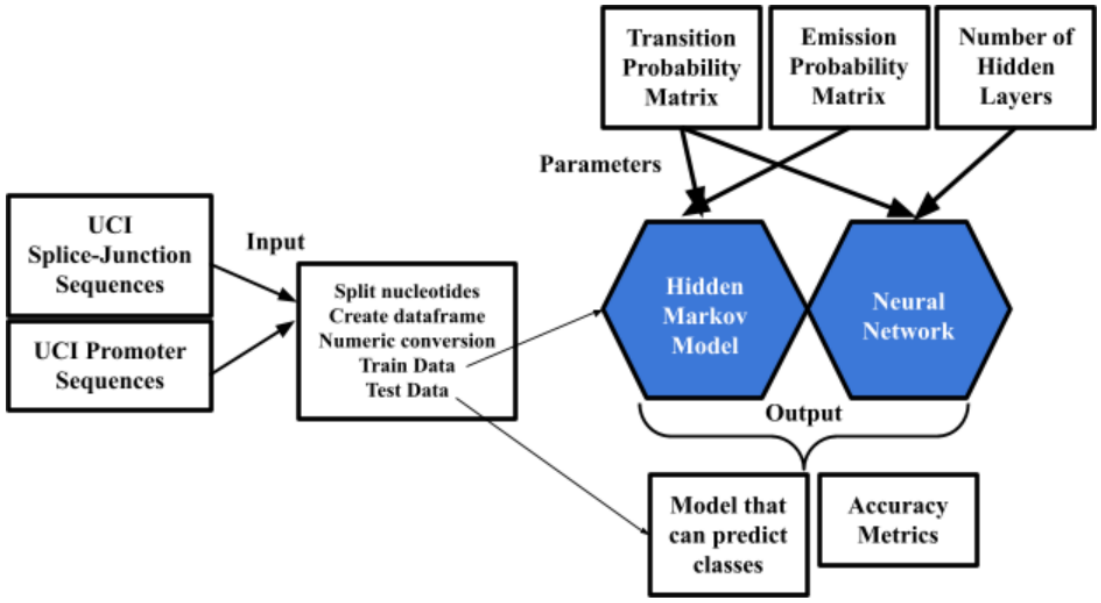
3.1. Flowchart of the originally proposed approach



3.2. Proposed approach / analysis plan

Software packages such as numpy, pandas, and sklearn will be crucial to my project. They provide methods for data manipulation, analysis, and baseline steps of machine learning. The Kaggle tutorial provided an example of how to use these libraries to load in a dataset and create suitable inputs for the algorithms.

One analytical method I will use is just a plain comparison of the accuracy metrics for the different models. This is a preliminary result that allows for an evaluation without statistical confirmation. However, I will then accumulate average values for multiple runs of each algorithm and use a statistical test, such as a t-test, to see if there is a significant difference between the mean accuracy of the two different algorithms run on the same dataset.



3.3. Approach taken

Step 1: Prepare data for algorithms

Goal: My goal was to create a dataframe that could be used by each algorithm.

Approach: For each dataset, I manipulated the shape/size, data types, and column headers. I also split the datasets into training and testing data.

Outcome:

	0_a	0_c	0_g	0_t	1_a	1_c	1_g	1_t	2_a	2_c	...	54_t	55_a	55_c	55_g	55_t	56_a	56_c	56_g	56_t	Class
0	0	0	0	1	1	0	0	0	0	1	...	0	0	0	1	0	0	0	0	1	1
1	0	0	0	1	0	0	1	0	0	1	...	0	1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	0	1	1	0	...	0	0	1	0	0	0	0	1	0	1
3	1	0	0	0	1	0	0	0	0	0	...	0	0	0	0	1	0	1	0	0	1
4	0	0	0	1	0	1	0	0	0	0	...	1	1	0	0	0	0	0	1	0	1

Figure 1. Fully Processed Promoter Sequences Dataset

Learnings: Even datasets from the same website have nuances that have to be accounted for. It can be a trial and error process at first to ensure the sizes of matrices and rows and columns match up.

Step 2: Run well-known predictive algorithms on each dataset

Goal: I wanted to get a sense for the range of predictive abilities of each algorithm, and ensure the data was processed and split into training and testing sets successfully.

Approach: I ran Nearest Neighbors, Decision Tree, Random Forest, Neural Net, AdaBoost, Naive Bayes, SVM Linear, SVM RBF, and SVM Sigmoid models on the same training and testing data. I adapted code from Neural Networks Demystified[7] and hmmlearn[5] to implement the ANN and HMM.

Outcome:

Dataset	Hidden Markov Model	Artificial Neural Network	Nearest Neighbor	SVM Linear	Decision Tree	Random Forest	Neural Net	AdaBoost
UCI Promoter	0.49	0.997	0.78	0.96	0.78	0.67	0.926	0.85
UCI Splice-Junctions	?	0.98	0.88	0.97	0.98	0.76	0.984	0.984

Figure 2. Table of Testing Accuracy Values for the Different Algorithms.

Learnings: The figure above shows the testing accuracy values for 8 different predictive algorithms. The algorithms were trained on the training data (75%) and then run on the testing data. The predictions were compared to the actual classes and the sklearn accuracy score function was used to discern the above values.

Certain algorithms performed better across the board, for instance the Artificial Neural Network and SVM Linear algorithms were both in the high 90s for both datasets. However, between datasets there were also some discrepancies. Decision tree and AdaBoost performed much better on the Splice-Junction dataset than the promoter dataset. This is likely based on the patterns in the data and the ability of the algorithms to fit the data and classes in a limited number of iterations.

Step 3: Compare time and accuracy values

Goal: My goal was to determine the merits of each algorithm by averaging the calculated elapsed time and accuracy for each run.

Approach: I created a loop that iterated a certain number of times and trained the ANN, while keeping track of elapsed time. For each algorithm, the number of hidden layers was also varied and tracked. Then the accuracy of the training and testing predictions were determined. Finally, the values for corresponding scenarios were averaged and plotted. I also fitted the HMM and kept track of elapsed time

Outcome: minimum elapsed time for an ANN run was 5.35 seconds, and the maximum was 71.3 seconds. Although the HMM fitting had some bugs and was very poorly trained, it ran in about 1.5 seconds.

Learnings: Accuracy and run time were not always correlated.Both were more determined by the goodness of fit of the complexity of the model.

4. Summary of key findings and learnings

DNA seems to be best classified by ANN

Dataset	Hidden Markov Model	Artificial Neural Network	Nearest Neighbor	SVM Linear	Decision Tree	Random Forest	Neural Net	AdaBoost
Both datasets	0.49	0.989	0.83	0.965	0.88	0.715	0.955	0.917

Figure 3. Table of Testing Accuracy Values for the Different Algorithms.

The figure above shows the average testing accuracy values for 8 different predictive algorithms. The algorithm that performed the best on determining the correct classification for a sequence of DNA was the Artificial Neural Network.

Clearly, performance varies between datasets, so this pattern cannot be extrapolated to all DNA, genes, and sequences. However, it is likely based on the adaptability of learning algorithms that an ANN would continue to perform well. Additionally, if I had been able to better train the Hidden Markov Model, and perhaps create a more accurate starting transition probability matrix, the accuracy of that model could have been much higher. In the literature I read, HMMs performed very well with gene prediction and classification - hence why I chose to work with it.

Parameters impact model performance

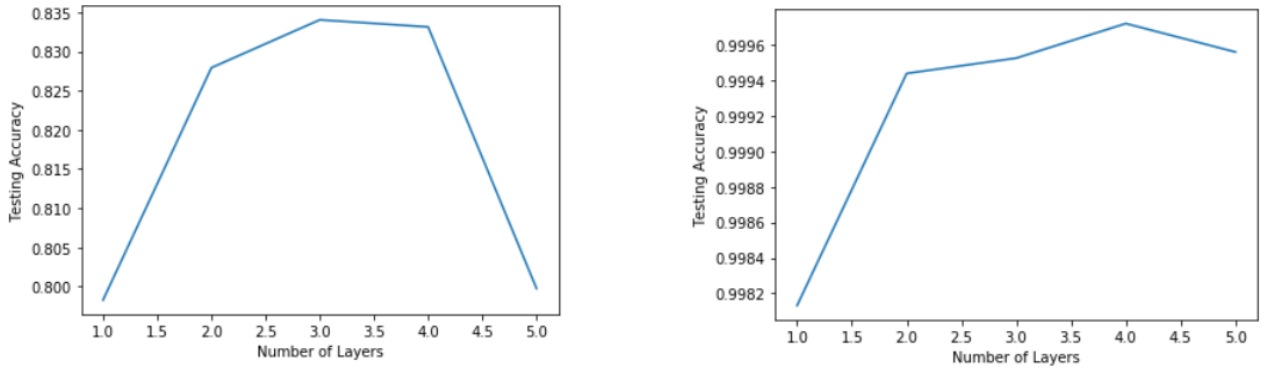


Figure 4. Promoter(left) and Splice-Junction(right) sequence average ANN testing accuracies for varying layers

To confirm that the accuracy values I calculated for each number of layers (1, 2, 3, 4, 5) were accurate, I ran each 10 times and then averaged the values. I then plotted the averages as shown above.

In the figure above, it is clear that an ANN with 3 hidden layers performs best at prediction on average, while ANNs with 1 and 5 layers perform the worst. Note the scale of the y-axis however ranges from 0.8 to 0.83, so the difference between the accuracy values is not enormous. Still, these differences show that an ANN with only 1 layer does not have enough detail to capture the patterns in the data while 5 layers introduce too much complexity which becomes detrimental.

Time is a significant constraint

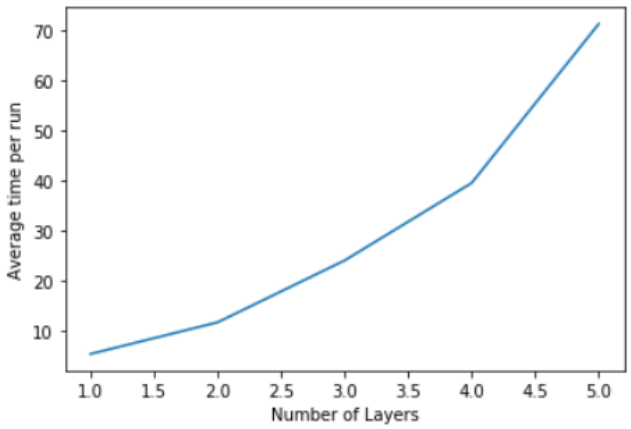


Figure 5. Plot of average time per run in seconds for the ANN

As I was running the ANN for each number of layers 10 times, I also used a python timing function to keep track of how long each was taking. The graph shows that the ANN runs in around O(n) time where n is the number of hidden layers. For instance, the average run time for an ANN with 5 layers was 70 seconds. When that is being done 10 times in addition to all the other numbers of layers, the time really adds up. I had to be very patient and realize that there wasn't an infinite loop or error in the code - it just took a long while, 25 minutes to run the one block of code.

Originally, I planned to run each ANN with a certain number of layers 20 times, but this evaluated to 100 instances of creating, training, and forwarding the ANN. As shown in the figure above, this was just not feasible given the amount of time each run took. If I was using a software platform other than Jupyter Notebook that allowed me to run a program, leave it for a few hours, and then come back to the data, time would not have been as big a limiting factor.

Code and data availability

This code is available at the github repository dna-classification-project created by perryk12. **https://github.com/perryk12/dna-classification-project**

I was familiar with github coming into this project, so I found creating the repository and pushing commits and files to it via command line relatively easy. The main code is organized into two jupyter notebooks, which are located in the results folder. The first performs analysis of

the Promoter Sequences dataset, the second the Splice Junction Sequences dataset. Both can be downloaded by anyone and reproduced by running the cells consecutively. The code will load in each dataset by archived url, but in the event that the data at that location disappears, the src folder has each data file saved. Also in the src folder is a file dependencies.sh which is a script that can be run to install the packages used in this project (such as numpy, matplotlib, pandas, scikit-learn, and hmmlearn[4]).

5. Challenges

Comments on specific results/outcomes/milestones set at the beginning

I reached the majority of milestones that I set. For instance, I successfully loaded all my datasets, ran the code for the HMM and ANN, which were no easy feat. The one milestone I didn't fully achieve was having well-trained models for both datasets. My ANNs were well-trained but the HMM was not.

Technical/scientific challenges faced

One challenge I faced was that implementing a Hidden Markov Model was a lot more difficult that I expected. I ran into issues with importing the package at first, and then later faced usage difficulties because the documentation did not make it clear how to input vectors and labels into the function to fit the model. After finally ensuring the shapes and datatypes of the inputs were correct, the code to fit and predict ran successfully but the predictions had an accuracy value of about 50%, which given the almost binary array of output values, shows that it was pretty random.

Practical challenges faced

The long runtime of some of the algorithms was an obstacle at times. At one point I thought the code just wasn't going to finish and when I left for a few minutes, the jupyter notebook connection had been lost. As I mention in the next section, Jupyter Notebook presented additional challenges when I had to rerun the data input and algorithms every time, and the accuracy scores would vary and sometimes drop unexpectedly.

6. Reflection and future directions

If I were to start this project from scratch, I would develop a more focused problem - such as investigating how the length of sequences or nucleotide ratios affect the classification and prediction abilities of an ANN and HMM. I would also download the datasets as files on my computer instead of loading it from the url each time. That would enable certainty of the consistency of the data. I think that Jupyter Notebook is a great tool for code development that allows for compartmentalization and quick running and rerunning of code; however, I would change my approach by saving the variable values and processed dataset somehow to a file so that I could load them in instead of recalculating.

In the future, this project could be extended by creating a better hidden markov model based on the transition and emission probability matrices. It would be intriguing to calculate these matrices based on the data from running BLAST on the sequences. The probabilities of the matching sequences and genomes could be used in the matrices as a way to integrate the extrinsic evidence mentioned in the Problem and Goals section.

7. Acknowledgements

I would like to thank Matthew Andres Moreno for aiding me on this project by answering questions and guiding me on ideas and project directions.
I would also like to thank Arjun Krishnan for teaching me about the topics of this project in addition to offering insight and feedback on my project approach and methods.

8. References

[1] Bulentsiyah. "Classifying DNA Sequences-Markov Models-KNN-SVM." *Kaggle*, 17 May 2020,www.kaggle.com/bulentsiyah/classifying-dna-sequences-Markov-models-knn-svm/execution

[2] D. V. Lindberg and H. Omre, "Inference of the Transition Matrix in Convolved Hidden Markov Models and the Generalized Baum–Welch Algorithm," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 12, pp. 6443-6456, Dec. 2015, https://ieeexplore-ieee-org.proxy2.cl.msu.edu/document/7147792, doi: 10.1109/TGRS.2015.2440415.

[3] Grewal, Jasleen K., et al. "Markov Models -- training and evaluation of Hidden Markov Models." *Nature Methods*, vol. 17, no. 2, 2020, p. 121+. *Gale OneFile: Health and Medicine*, https://go-gale-com.proxy2.cl.msu.edu/ps/i.do?p=HRCA&u=msu_main&id=GALE%7CA613230415&v=2.1&it=r&sid=summon Accessed 12 Feb. 2021.

[4] Hmmlearn. "Hmmlearn." *GitHub*, 5 Feb. 2021, github.com/hmmlearn/hmmlearn.

[5] Noordewier, Michiel, et al. "Training Knowledge-Based Neural Networks to Recognize Genes in DNA Sequences." *Proceedings of the 3rd International Conference on Neural Information Processing Systems*, 1990, pp. 530–36, proceedings.neurips.cc/paper/1990/file/8efb100a295c0c690931222ff4467bb8-Paper.pdf.

[6] Wang, Zhuo, et al. "A Brief Review of Computational Gene Prediction Methods." *Genomics, Proteomics & Bioinformatics*, vol. 2, no. 4, 2004, pp. 216–221., https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5187414/#:~:text=There%20are%20mainly%20two%20classes,as%20ab%20initio%20gene%20finding. doi:10.1016/s1672-0229(04)02028-5.

[7] Welch, Stephen. "Neural-Networks-Demystified." *GitHub*, 9 Mar. 2020, github.com/stephencwelch/Neural-Networks-Demystified.

9. Glossary

DNA classification: grouping or identifying genetic sequences based on shared characteristics or functionality
Gene prediction: identifying segments of DNA that encode genes
Hidden Markov Model: models data using hidden states and transition probabilities
Transition probability matrix: a matrix of the probabilities of changing from one state to another
Emission probability matrix: a matrix of the output probabilities, likelihood to observe a variable value given a particular state
Neural Network: a program with layers based on a brain that can be trained to predict labels
Intrinsic evidence: data that is directly within the set of sequences
Extrinsic evidence: data that is not within the set of sequences, context
BLAST: A tool to create local sequence alignments of dna sequences
AUGUSTUS: program that uses a Generalized Hidden Markov Model to do gene prediction
Promoter: region of dna where proteins bind to start the process of transcription
Splice-Junction: the location where an intron was spliced out to create processed mRNA