# BIEN 410 Project

## Abstract

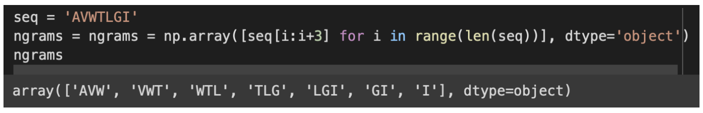
Fast and reliable protein secondary structure prediction is desirable when an amino acid sequence structure has not been resolved using techniques such as X-ray crystallography, nuclear magnetic resonance spectroscopy, or cryo-electron microscopy. This report evaluated the two-state protein secondary structure prediction accuracy of non-parametric, probabilistic, and deep learning methods on a labelled dataset of 5326 FASTA sequences. The models evaluated included a k-Nearest Neighbour Classifier (KNN), Categorical Naive Bayes Classifier (CNB), Hidden Markov Model (HMM), and a Recurrent Neural Network (RNN). Techniques from the field of Natural Language Processing (NLP), such as the use of an n-gram approach to sequence representation, were also applied. An n-gram sequence representation improved validation accuracy across all models except the HMM, which cannot use an n-gram input. Furthermore, the RNN achieved a validation accuracy of 72%, the highest among the models. The results of this work suggest that models and pre-processing techniques from the field of NLP may be well suited to protein structure prediction. However, the RNN model also presents limitations due to the large size of the model parameters and the requirement of equal-length input sequences.

## 1 Code Description and Motivation

### 1.1 Motivation

The Categorical Naive Bayes Classifier is a surprisingly powerful classification algorithm even when the conditional independence assumption does not hold [Domingos and Pazzani, 1997]. Therefore, it served as a baseline for comparing other methods. The RNN and HMM were explicitly selected for their utility in modelling sequential data, whereas the KNN served as the simplest non-parametric model for classification [Murphy, 2022]. Recent work on protein structure prediction has leveraged n-gram representations from NLP [Islam et al., 2018], which treat the protein sequence as a text string and use an n-sized sliding window with a stride of 1 to create the representation (1). Here, a 3-gram representation of the sequences expanded the "corpus" to perform tokenization from 20 to 8414, enabling richer feature representation with greater context relative to tokenizing amino acids only.

Several n-gram sizes were tested with the CNB classifier to examine the relationship between n and validation accuracy (2). Here, 25% of the available data was selected for validation, meaning that it was not used in calculating the log-prior and log-likelihood probabilities. An accuracy asymptote of around 69.5% is reached with the CNB classifier at N=9; however, other research has suggested that an n-gram size of approximately 4 is most appropriate for protein sequences [Maetschke et al., 2010]. Furthermore, a larger n may result in more overfitting to the training set, and a larger generalization error [Murphy, 2022]. Considering the potential of overfitting along with the limited file size of "parameters.txt", an n-gram size of 3 was chosen to compare all models (except the HMM, where an n-gram input representation is not applicable).



```
seq = 'AVWTLGI'
ngrams = ngrams = np.array([seq[i:i+3] for i in range(len(seq))], dtype='object')
ngrams

array(['AVW', 'VWT', 'WTL', 'TLG', 'LGI', 'GI', 'I'], dtype=object)
```

Figure 1: Example of 3-gram processing of an encoded sequence. Here a sliding window of size 3 is used with a stride of 1

Validation accuracy, prediction time, and input data pre-processing requirements were considered during the model comparison. All tests were performed using the same sample of validation data (25% of the total data) and 3-gram feature representation (other than the HMM). Regarding accuracy, the HMM, CNB, and KNN (with K=90) had validation accuracies of 65%, 66.5% and 67%, respectively. Of the three models, inference with the CNB classifier was the fastest. Comparing the inference time complexities, the HMM using the Viterbi algorithm is $\mathcal{O}(NC^2T)$, the CNB classifier is $\mathcal{O}(NCD)$ and the KNN is $\mathcal{O}(NKD)$ for N sequences, C classes, T time steps, D features, and K number of neighbours [Chatterjee and Russell, 2012]. Therefore, when hundreds or thousands of amino acids exist in each sequence, inference with the HMM is significantly slower than the other two models.

The RNN model with LSTM units achieved the highest validation accuracy of 72%. It is also relatively fast at prediction; however, due to memory constraints, batching sequences during inference is not feasible with this NumPy implementation, so its inference speed is reduced while performing one-by-one predictions. Another drawback of the RNN is that input sequences must be the same length. To meet this requirement, sequences must be truncated or padded to a specific length. Given these considerations, the RNN's superior performance and relatively fast inference made it the model of choice for this project.

### 1.2 Data Pre-Processing

Data pre-processing is initialized by the readInput() function of SSpred.py, which is unmodified from the original SSpred.py example file and returns a dictionary, inputData, with the values containing the string sequences from the input FASTA text file. During prediction, each sequence from inputData is pre-processed using the PreprocessorLSTM class and calling the fit() method on a sequence string as input. The fit() method first creates a 3-gram representation of the input sequence through a sliding window ap-

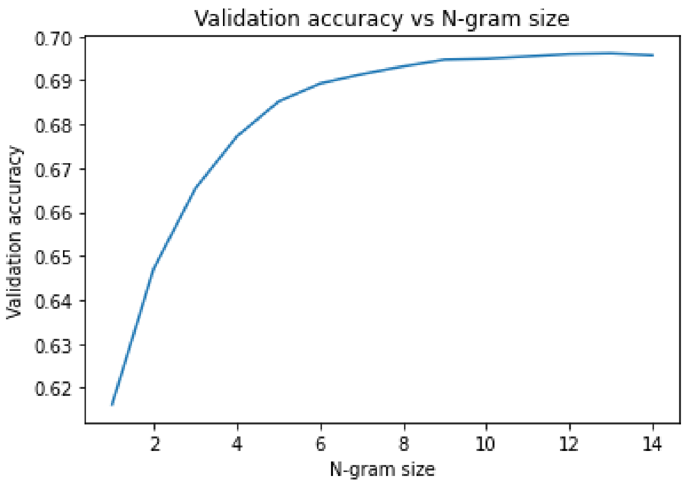


Figure 2: Validation accuracy using 25% of available data vs. N-gram size for a CNB classifier. An asymptote of ≈ 69.5% is reached with N = 9

proach of size 3 with a stride of 1. After splitting the sequence into 3-grams, the 3-grams are numerically tokenized using a dictionary of all possible 3-gram tokens seen during training on the entire training set. Next, the sequence is padded with zeros (which do not correspond to any 3-gram) or truncated to a length of 1419. This length was chosen since it was the maximum sequence length in the training set. After these steps, the tokenized 3-gram sequence representation of length 1419 can be used during inference.

### 1.3 Training

The RNN model was trained with 5326 training sequences after pre-processing by padding, tokenizing, and representing in a 3-gram form as described in Data Pre-Processing. Similarly, the labels were zero-padded to a length of 1419, tokenized as either 0, 1, or 2 and one-hot-encoded in vectors of length 3. The training was performed with an Adam optimizer, a learning rate of 0.08 and a categorical cross-entropy loss function, which is convex with respect to the weights during training [Murphy, 2022]. Here, the machine learning library Keras was used to automatically differentiate the weights and update them via gradient descent as described in RNN. A validation set was used to monitor accuracy to assess overfitting during training.

### 1.4 Inference

When SSpred.py is run, the input data is read by the readInput() function as described above. The inputData dictionary returned from readInput(), along with the parameters.txt file, are input parameters for the predict() function. Within predict(), each sequence in inputData is pre-processed using a 3-gram tokenizing dictionary and passed to the predictLSTM() function, which performs a forward pass through the model layers using the saved model weights from training. The model prediction is then converted from a one-hot representation to a string and stored as a value in the my_prediction dictionary with the pre-amble from inputData as the key. Finally, The my_prediction dictionary, inputData and the file path for the outfile are passed to the writeOutput() function, which is unmodified from the original example and writes the predictions to outfile.txt.

## 2 Theoretical Basis

### 2.1 KNN

k-Nearest Neighbors is a non-parametric supervised learning algorithm used for classification and regression in machine learning. This project used a KNN classifier with a Hamming distance function since the input features are categorical [Murphy, 2022]. KNN aims to estimate an unknown data point based on its nearest neighbours denoted by the symbol "k" (k refers to the number of neighbours selected). k = 90 here was selected since it provided the best accuracy on the validation data. During KNN training, all training data points (in this case, all tokenized 3-gram representations of sequences) are stored. The k-nearest neighbours are determined using the Hamming distance function during the inference of a single 3-gram. For each k-nearest neighbour, the label class is recorded, and the most frequent class is selected as the predicted class for the given 3-gram [Murphy, 2022].

### 2.2 HMM

A Hidden Markov Model is based on a Markov Chain. A Markov chain tells us about the probabilities of sequences of "states" and relies on the assumption that all that matters for predicting the next state of a sequence is the current state. The "hidden" part of the HMM refers to the fact that the events we care about (in this case, predicting H or "-") are not observed. Instead, we observe the protein sequence and want to predict the most likely hidden sequence of states for the observed protein sequence. The way to do that using dynamic programming involves calculating the initial probability distribution of the states, the transition probability matrix for the states and the emission matrix for the observed sequence. We can use the Viterbi algorithm with these parameters, which fills out each cell in the HMM dynamic programming matrix we learned in class by taking the most probable path that could lead to a given cell in the matrix [Keselj, 2009].

### 2.3 CNB

Categorical Naive Bayes is an extension from Naive Bayes that we learned in class where each of the input features ($x_1$, $x_2$, $x_3$ for the 3-gram encoding) could take on multiple values (in this case 20 for the number of amino acids). CNB applies a strong conditional independence assumption of the input features with Bayes Theorem to classify a new example. More formally, given input features $\mathrm{x}_1$, $\mathrm{x}_2$, $\mathrm{x}_3$, Naive Bayes tells us: $p(c|x_1, x_2, x_3) = p(c)p(x_1|c)p(x_2|c)p(x_3|c)/p(x_1, x_2, x_3)$. Therefore, each of these terms can be computed by counting for each class the fraction of each possible feature value of $x_1, x_2, x_3$. The most probable class for a given set of features is the predicted class for a given 3-gram.

### 2.4 RNN

A Recurrent Neural Network maps an input space to an output space in a stateful way, meaning that an output $\mathrm{y}_t$ depends not only on the input $\mathrm{x}_t$ but also the hidden state $\mathrm{h}_t$ which is developed across the timesteps of a sequence during model training [Murphy, 2022]. The RNN model trained here is part of a class of aligned Seq2Seq models which deal with sequence translation. The model architecture (3) takes in an input sequence padded or truncated to a length of 1419 and performs embedding of that sequence which maps each 3 gram of that sequence to a learned embedding vector of length 4. The resulting output is of shape 1419 by 4. The weights for each of these layers (embedding, LSTM, and dense) are learned through gradient descent; however, unlike gradient descent for logistic regression, which we discussed in class, the derivative of the loss function with respect to the weights for each layer is calculated using backpropagation. For the gradient calculation, backpropagation recursively applies the chain rule [Murphy, 2022].

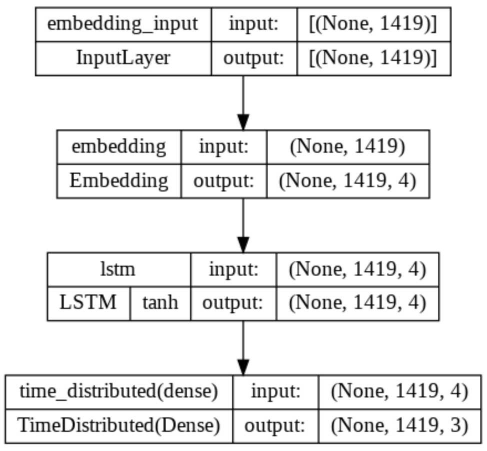


Figure 3: Illustration of the RNN Model used

Next, the sequence is passed to an LSTM Layer with 4 LSTM units (4). The sequence is passed through each LSTM unit (giving an output of 1419 by 1). Then, the outputs of the 4 LSTM units are stacked for an output of 1419 by 4. The stacked output of the LSTM layer is fed into a fully connected layer with three output units corresponding to the two classes “H” and “-”, plus a class for zero-padding. Finally, a softmax function is applied to the fully connected layer outputs such that the value for each of the three classes can be interpreted as a probability. The class with the highest probability is chosen as the predicted category for each specific 3-gram input.

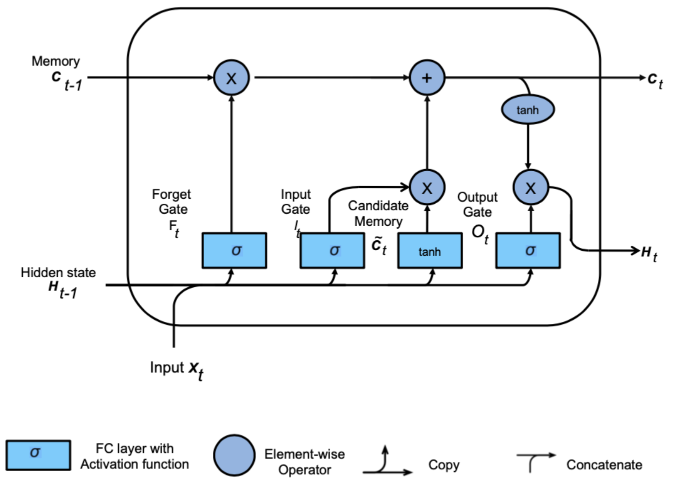


Figure 4: Illustration of a long short term memory unit adapted from [Zhang et al., 2021]. Here the hidden state $\mathrm{h}_t$ is controlled by the previous hidden state $\mathrm{h}_{t-1}$, the current input $\mathrm{x}_t$, and the previous memory cell state $\mathrm{c}_{t-1}$. This cell seeks to maintain a stateful representation of the sequence and selectively “remember” important information, or reset the hidden state to “forget” things that are no longer useful.

## 3 Advantages and Disadvantages of This Approach

Many of the advantages and disadvantages mentioned in Motivation lead to the choice of the RNN model and 3-gram encoding. In summary, these advantages included a $> 2\%$ improvement in validation accuracy over a CNB classifier and leveraging of the strong correlation of n-gram patterns and secondary structure, specifically for n = 4 [Vries et al., 2007]. The main disadvantages for the RNN include the use of padding and truncating since this limits the size of the input sequence and, in the case of padding, can negatively affect model performance [Dwarampudi and Reddy, 2019]. In addition, due to the parameters.txt file size limit of 500kb, an n-gram size of four could not be used despite previous research showing that four is likely the optimal n-gram size for protein secondary structure prediction.

## 4 Suggested Improvements

There are two ways to overcome the disadvantages of this approach. First, masking of the padding should be applied to the layers of the RNN during training and inference and second, an n-gram representation of size four should be used. When masking is used, the zero-padded time steps are "skipped" during the model training, meaning that the loss calculation does not depend on the padding. As a result, the model performance is likely to improve since it is not optimizing the prediction of padding (which constitutes most of the data in this case). From the model's confusion matrix (5), we can see that most of the sequences are padding, and there are misclassifications of "H" and "-" as zero padding. To use n-grams of size four, the size limit of parameters.txt would need to be increased to accommodate for the larger embedding weight matrix.

Regarding model design, further improvements could include using more advanced NLP models such as transformers which leverage concepts such as attention and positional encoding to understand better the context of input sequences [Lauriola and Moschitti, 2020].

## 5 Conclusion

The RNN model adapted from NLP Seq2Seq tasks improved protein secondary structure prediction accuracy relative to non-parametric and probabilistic models. Using a CNB classifier for baseline experiments, it was found that n-gram encoding improved validation accuracy, which was observed across all models (except HMM, where such a representation is not applicable). A validation accuracy of 72% was achieved with the RNN model presented here using a 3-gram encoding. Further work should consider incorporating more advanced NLP models, such as transformers and masking during training and inference, to avoid padding influencing the optimization during backpropagation.

## 6 Additional Note

All model training was performed in Google Colab which enabled the use of GPUs to train the RNN. To see how the KNN, HMM, RNN, and LSTM were trained, I included a *BIEN410_TrainingScripts.ipynb* jupyter notebook file. This file can be uploaded to Google Colab and be easily run by specifying the path to the project folder, infile.txt, and labels.txt within a Google drive under the "Import Data" heading.

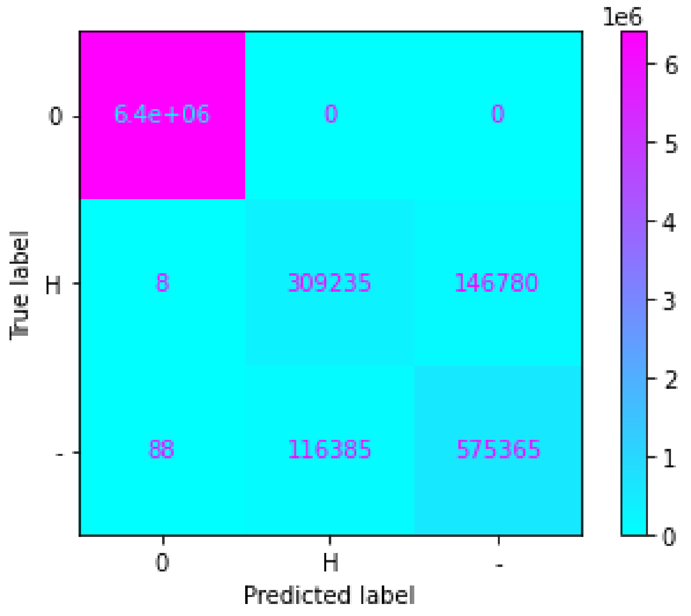


Figure 5: Confusion matrix of the trained model on the entire training set.

## References

Shaunak Chatterjee and Stuart Russell. A temporally abstracted viterbi algorithm. *arXiv preprint arXiv:1202.3707*, 2012.

Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2):103–130, 1997.

Mahidhar Dwarampudi and NV Reddy. Effects of padding on lstms and cnns. *arXiv preprint arXiv:1903.07288*, 2019.

SM Ashiqul Islam, Benjamin J Heil, Christopher Michel Kearney, and Erich J Baker. Protein classification using modified n-grams and skip-grams. *Bioinformatics*, 34(9):1481–1487, 2018.

Vlado Keselj. Speech and language processing daniel jurafsky and james h. martin (stanford university and university of colorado at boulder) pearson prentice hall, 2009, xxxi+ 988 pp; hardbound, isbn 978-0-13-187321-6, 2009.

Ivano Lauriola and Alessandro Moschitti. Context-based transformer models for answer sentence selection. *arXiv preprint arXiv:2006.01285*, 2020.

Stefan R Maetschke, Karin S Kassahn, Jasmyn A Dunn, Siew-Ping Han, Eva Z Curley, Katryn J Stacey, and Mark A Ragan. A visual framework for sequence analysis using n-grams and spectral rearrangement. *Bioinformatics*, 26(6):737–744, 2010.

Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

John K Vries, Xiong Liu, and Ivet Bahar. The relationship between n-gram patterns and protein secondary structure. *Proteins: Structure, Function, and Bioinformatics*, 68(4):830–838, 2007.

Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.