MASTER OF SCIENCE
IN ENGINEERING

Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Orientation : Technologies de l'information et de la communication (TIC)

Machine Learning on Genomics

Exploratory Research on Natural Language Processing and Deep Learning techniques in Genomic Sequences Classification

Fait par

Bassem EL-KHANSAA

Lausanne, HES-SO//Master, Juillet 2021

Table of contents

I. Abstract ... *6*

II. Introduction ... *7*

i. Definitions ... **7**

ii. The project ... **8**

iii. CI4CB and Relevant Research ... **8**

III. Problem Statement ... *9*

IV. Objectives ... *10*

i. General Objectives ... **10**

ii. Specific Objectives ... **10**

V. Organization ... *11*

i. General Planning ... **11**

ii. Gannt ... **12**

VI. Methods ... *14*

i. Overview of the general approach ... **14**

ii. NLP classification methods ... **15**

VI.ii.1 Overview of DNA comparative analysis ... 15

VI.ii.2 BoW method - Alignment-Free approach ... 15

VI.ii.3 Information theory-based Alignment-Free methods ... 16

iii. Feature Selection Methods for NLP based classification ... **17**

iv. LSTM based classification methods. ... **18**

v. LSTM Sequence Encoding methods ... **18**

vi. Sequence Embedding. ... **19**

vii. Model evaluation methods ... **20**

VII. Study design ... *22*

VIII. Classification of Short Sequences ... *23*

i. Overview of the three Pipelines ... **23**

VIII.i.1 Data ... 25

ii. Classification with Bag of Words ... **30**

VIII.ii.1 Feature Extraction (BoW) ... 30

VIII.ii.2 Feature Selection, Tuning, and evaluation ... 33

iii. LSTM **48**
VIII.iii.1 LSTM Pipeline 48
VIII.iii.2 Training and evaluation 50
iv. LSTM on kmer encoded sequences. **52**
VIII.iv.1 LSTM on kmers Pipeline 52
VIII.iv.2 Training and evaluation 53
IX. Results, Conclusions and Discussions ***55***
X. References ***57***
XI. Annexes ***59***
XII. Source Code ***66***

Table of Figures

Figure 1 Genomics studies the genomes of whole organisms and other intragenomic interactions. Source: www.ebi.ac.uk 8
Figure 2 Gantt Chart of the project 12
Figure 3 Overlapping vs non-overlapping kmers. In the first, a window of size k is moving with step n<k. In the second, the window of size k is moving with step of size n = k 16
Figure 4 Best 20 features selected with an embedded Feature Selection method 22
Figure 5 General Pipeline of the experiment 24
Figure 6 An example of a multi-FASTA file 25
Figure 7 Shows how I split a multi-fasta file into multiple single fasta files. 26
Figure 8 Plot showing data distribution over the two classes 27
Figure 9 Histogram of sequences frequency 27
Figure 10 Box plot (left) ECDF plot (right) 28
Figure 11 Normality test 28
Figure 12 Feature extraction pipeline using Count Vectorizer 32
Figure 13 Effect of kmer length on Random Forest accuracy 34
Figure 14 Effect of kmer length on Logistic Regression accuracy 34
Figure 15 Effect of kmer length on SGD Classifier's accuracy 34
Figure 16 Feature importance score from Random Forest Classifier k=5. Left: pie chard. Right: Bar diagram. Both diagrams present the same information in two visually different ways, allowing for the reader's preference. 37
Figure 17 This graph shows performance on k=5 while searching for the best number of features, starting with the most important features, adding one feature at a time and repeating training. 37
Figure 18 AUC Score for Random Forest Classifier, k=5 37
Figure 19 Feature importance score from Random Forest Classifier k=6 38
Figure 20 This graph shows performance on k=6 while searching for the best number of features, starting with the most important features, adding one feature at a time and repeating training. 38
Figure 21 AUC Score for Random Forest Classifier, k=6 38
Figure 22 Feature importance score from Random Forest Classifier k=7 39
Figure 23 This graph shows performance on k=7 while searching for the best number of features, starting with the most important features, adding one feature at a time then repeating training. 39
Figure 24 AUC Score for Random Forest Classifier, k=6 + k=7 39
Figure 25 Feature importance from Random Forest Classifier k=6+7 41
Figure 26 Graph showing the performances with k=6 + k=7, of Random Forest Classifier w.r.t. total number of features. This search is used as a heuristic wrapper method to select the best number of features 41
Figure 27 AUC Score for RFC, k=6 + k=7 41
Figure 28 Validation curve of Random Forest model with respect to some hyperparameter 42
Figure 29 Performance of RFClassifier versus number of features. This is a sequential feature space search on a given range 43
Figure 30 AUC Score for RFC after tuning 44
Figure 31 Common patterns in the most important features (kmers) resulting from filter methods of design 2. 45
Figure 32 General LSTM Pipeline 49
Figure 33 LSTM deep model 50
Figure 34 LSTM on Kmer-encoded sequences pipeline. Here sequences are first padded, then kmer-encoded, before being fed into a deep LSTM model. 52

FIGURE 35 LSTM MODES, USED AS A CLASSIFIER AFTER FEATURE-SELECTION, TO ESTABLISH A BENCHMARK WITH CLASSICAL ML ALGORITHMS. 63
FIGURE 36 **TRAINING** AND **VALIDATION** ACCURACY OF LSTM ON KMERS WITH SIGMOID ACTIVAITON, OVER THREE EPOCHS. BOTH TRAINING AND VALIDATION ACCURACIES SWING AROUND 0.5. 64
FIGURE 37 EVALUATION (TESTING) **ACCURACY** OF LSTM ON KMERS WITH SIGMOID ACTIVAITON VERSUS NUMBER OF ITERATIONS, SWINGING AROUND 0.5........ 64
FIGURE 38 **TRAINING** AND **VALIDATION** LOSS, EXHIBITING SLIGHT DECREASE. 65
FIGURE 39 EVALUATION (TESTING) **LOSS** VERSUS NUMBER OF ITERATIONS, SWINGING AROUND 0.693........ 65

Acknowledgment

The thesis would not have been possible without the support of many valuable people. I would like to thank my supervisor, Prof. Carlos Peña-Reyes (CI4CB/HEIGVD), for his availability and guidance throughout this work. His guidance was necessary to get back on the right track whenever I drifted or got blocked. I would like to equally thank Ms. Shabnam Atae, for her valuable support, time, and comments. Many thanks to friends and family who supported me throughout the process. Thank you, mom, thank you dad for your support.

I am especially grateful to a very special person, for love and support throughout my studies, during these exceptionally hard times.

~When there is a will, there is a way~

I. Abstract

Genomics is the study of whole genomes of organisms and incorporates elements from genetics. Machine learning on genomics incorporate techniques of machine learning aiming to help genomic experts in their decision making.

In this applied research, we are interested in machine learning techniques most adapted for genomic sequence binary classification. Two main approaches are inspected: 1. Classical machine learning with methods used for Natural Language Processing, namely Bag of Words. 2. Deep learning, specifically LSTM, which is used in a fashion similar to the sentiment analysis.

II. Introduction

i. Definitions

Genomics: is the study of whole organisms' genomes. It differs from '*classical genetics*' in that it considers an organism's full genome, rather than one gene at a time. Moreover, genomics focuses on interactions between loci and alleles within the genome and other interactions such as epistasis, pleiotropy and heterosis (Figure 1).

Genome : Genome is all our DNA. From potatoes to puppies, all living organisms have their own genome. Each genome contains the information needed to build and maintain that organism throughout its life.[1] Knowing more about genome helps make informed decisions.

DNA: Genomes are made of DNA. DNA is read like a code. This code is made up of four types of chemical building blocks, adenine, thymine, cytosine, and guanine, abbreviated with the letters A, T, C, and G. The order of the letters in this code allows DNA to function in different ways.[2]

Double helix is the description of the structure of a DNA molecule. A DNA molecule consists of two strands that wind around each other like a twisted ladder. Each strand has a backbone made of alternating groups of sugar (deoxyribose) and phosphate groups. Attached to each sugar is one of four bases: adenine A, T, C, and G. The two strands are held together by bonds between the bases, A forming a base pair with T, and C forming a base pair with G.

Chromosomes: The DNA in a cell is divided into several segments of uneven lengths which can be tightly packed bundles known as chromosomes.

Genes: A gene is a segment of DNA, a region of DNA (deoxyribonucleic acid) coding either for the messenger RNA encoding the amino acid sequence in a polypeptide chain or for a functional RNA molecule. [3]

A Gene provides the cell with instructions for making a specific protein, which then carries out a particular function in the body. The differences in the letters within a gene are enough to change the shape and function of a protein, how much protein is made, when it is made, or where it is made.

.

[1] https://www.genome.gov

[2] https://www.genome.gov

[3] 2004 by Sinauer Associates, Inc. All rights reserved. Conner, J.K. and D.L. Hartl. A

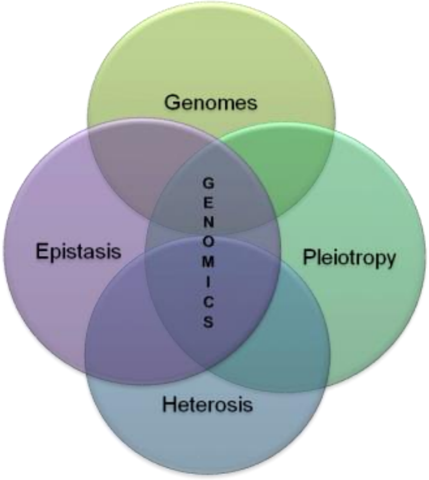


Figure 1 Genomics studies the genomes of whole organisms and other intragenomic interactions. Source: www.ebi.ac.uk

ii. The project

In the context of the master's graduation project, I chose to work on a topic in Machine Learning (ML) where I explore approaches for predictive modeling based on genomic information, i.e., DNA sequences (4 letter alphabet) or protein sequences (20 letter alphabet).

I join my efforts to those of the CI4CB team, working on the genomes of bacteria and viruses as part of the INPHINITY research project. One or more datasets from this project are to be provided.

This is an exploratory research project where I intend to rigorously show, whether the selected methods can achieve good results in classifying strands, in different settings with data of different complexities, and problems with various sizes.

iii. CI4CB and Relevant Research

ML on Genomics is a comprehensively researched topic by the CI4CB lab team at the HEIG-VD. The team of researchers is particularly interested in a track of ML methods that can be (but not necessarily) a starting point for my research.

Since the nature of genomic data is sequential, we may make hypothesis of similarities with text, which itself is of a sequential nature and since a group of genes together can have a meaning as a group of words together can have a meaning.

Their research track focuses on two main categories of ML methods that can be applied on genomics:

First. NLP (Natural Language Processing) on Genomics: which is a text classification method based on classical ML approach of Bag of Words. Its advantage is that it works well with small datasets, in terms of words to classify, and that is the case for genomics where sometimes we have a sparse amount of data.

Second. LSTM (Long Short-Term Memory) a deep learning approach usually used for text classification, speech recognition and anomaly detection in network traffic or IDSs (intrusion detection systems).

In these two approaches, we are interested in knowing whether we can apply these on genomics and obtain good results with various levels of size and complexity.

III. Problem Statement

As a preliminary research question, I put forward the following:

1. Which are recent ML methods that are adapted to the genomics?
2. Which of these methods seem to be most promising?
3. Which of these methods can be implemented?
4. Is the method adapted to the type of problem in question?
5. For which dataset size is a method designed to work on?
6. Is it performant on bigger dataset sizes?
7. How does the method perform with bigger size of the data? From an order of 10^3 positions (for Viruses) up to 10^7 positions (for Bacteria).
8. How does the method behave with mixed data types? (a mix of viruses plus bacteria DNA sequences as input).

IV. Objectives

i. General Objectives

The project will consist of the following general objectives.

1. Establishing an inventory of machine learning techniques suitable for processing genomic sequences,
2. Selecting and implementing (at least) one method that meets the specific needs of the project. Needs that will be communicated during the initial discussion.
3. Test the method on data sets of increasing complexity and difficulty. Evaluate it in terms of the predictive power, and computation cost and time.
4. Documentation

ii. Specific Objectives

The specific objectives are to be achieved in order. The validation of the one is required to proceed with the next one. They are summarized as follows:

1. Classification of short artificially generated DNA sequences. The training set consists of 60,000 artificially generated sequences of length between 5k-15k character. The total data size of the FASTA file is ~600MB.
2. Classification of real DNA Viruses and Bacteria DNA sequences. The training set will consist of much longer sequences than the artificial dataset. At this level, I may be able to test the robustness and scale-up capacity of the models generated at the first objective.
3. Classification of a DNA sequences of mixed nature, with higher complexity. Details on this objective are to be communicated if the second objective is met.

V. Organization

i. General Planning

The following planning represents the projected timeline of the project. These are estimations and the actual time needed for a task depends on the complexity as well as unexpected technical problems. This planning serves as a reference to measure the progress towards the objectives set ahead.

Weeks 1 to 2 – Preparation & Research Proposal

1. First meeting with project stakeholders.
2. Revision of ML theories.
3. Research on Genomics.
   a. EMBL-EBI
   b. Genome.gov
   c. ebi.ac.uk
4. Compiling a list of potentially pertinent articles for literature review
5. Elaboration of the Research Proposal

Weeks 3 to 6 – Literature Review

1. Searching for a similar and recent work (inventory of ML methods on Genomics)
2. Spotting and compiling potential papers and sources.
3. Defining nomination/elimination criteria.
4. Filtering of relevant research following criteria.
5. Updating objectives
6. Evolve the planning.

Weeks 7 to 12 – Implementation and adaptation of selected Methods for classification on short sequence.

1. Conception of preprocessing pipeline
2. Implementation of the pipeline methods
3. Iteratively adapting/modifying the methods or completely replacing the methods
4. Choosing adequate classification algorithms
5. Training and evaluation on preprocessed data
6. Reporting and documenting the results.

Weeks 12 to 14 – classification on viral sequences.

Weeks 14 to 16 – classification on bacterial sequences

Weeks 16 to 18 – classification on heterogeneous sequences

Week 18-20 Finalizing the report

ii. Gannt

To have a visual representation of the project timeline, a Gannt diagram is generated based on the planning estimates. It is frequently consulted for a better global evaluation of the overall progress towards the objectives. The timeline is flexible and is adapted based on the real achievements and time taken to achieve a milestone.

Below is a preliminary Gannt diagram that represents the different tasks and their interdependency.

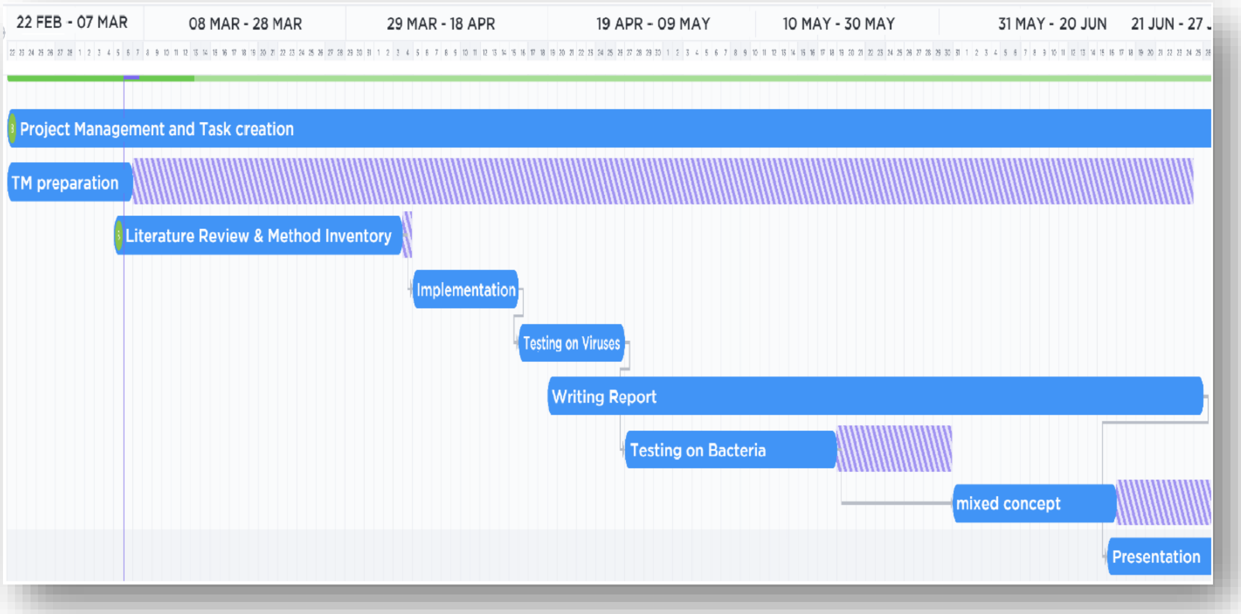


Figure 2 Gantt Chart of the project

Methods

VI. Methods

i. Overview of the general approach

In this chapter I introduce three approaches for comparative analysis of DNA. The first is based on Natural Language Processing (NLP), while the second and the third are based on deep learning (DL).

The main challenges in genomic sequence classification reside in the preprocessing, feature extraction, and feature selection processes. The absence of clear features and the explosion of dimensionality make the task very challenging, and the current difficulty of research resides in how to effectively represent sequence features and analyze high dimensional data *(Yang et al. 2020).* This makes it clear that the bottleneck in this project would be at this phase, the data preprocessing and feature extraction for the Natural Language Processing approach.

However, this does not apply on the Deep Learning based approach, which inherently does not require feature extraction. Yet, some preprocessing would be needed for the Deep Learning to optimize hardware resources usage, especially for the third approach, where sequences need to be encoded.

Approach 1> Natural Language Processing (NLP) based comparative analysis.

Classification tasks constitute a main building block of Indexing techniques for genomic sequences, answering the questions as for to which class or category does a sequence belong (i.e., taking a sequence as input and giving a predicted class as output). For this purpose, I am going to present two main approaches dealing with Comparative analysis of DNA, namely, Alignment-Based and Alignment-Free methods. In this research, we focus on AF approaches that allows the use of Bag-of-Words model mainly used in NLP.

Approach 2> Deep Learning (DL) using LSTM.

Comparative-analysis methods are not a domain-specific and/or knowledge-based methods. Extracting relevant features from DNA sequences can be a non-trivial task by itself. Even if these methods work on certain problems, there is no guarantee for generalization on a similar problem with different dataset.

Deep Learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input(Wikipedia contributors 2021b), allowing us to skip the feature extraction step.

We are interested in studying a specific sub-class called Recurrent Neural Networks (RNN), specifically the Long Short-Term Memory (LSTM) architecture.

Approach 3> LSTM on kmer encoded sequences.

A third approach inspected in this research, is the application of LSTM on sequences encoded as kmers. A preprocessing step needs to be taken I order to prepare the sequences for the training. This encodes the sequences into Kmers, separated by a special character, making sequences of kmers similar to sentences of words.

The implementation of these three approaches requires various methods, which will be discussed in this chapter.

ii. NLP classification methods

VI.ii.1 Overview of DNA comparative analysis

VI.ii.1.1 SA (Sequence alignment)-based approach:

In a pairwise alignment (i.e., the alignment of two sequences), we are given two sequences A and B and are to find their best alignment (either global or local). DNA sequences alphabet is the 4-letter set {A, C, G, T} and protein sequences alphabet is the 20-letter set {A, C−I, K−N, P−T, V WY}. Alignment-based approaches generally remain the references for sequence comparison, however, SA-based methods do not scale with the very large data sets that are available today. In addition, aligning two long DNA sequences is infeasible in practice (Zielezinski et al. 2019). Therefore, as an alternative to sequence alignment, many so-called alignment-free (AF) approaches to sequence analysis have been developed, with the earliest works dating back to the mid-1970s. Although the concept of the alignment-independent sequence comparison gained increased attention only in the beginning of the 2000s.

VI.ii.1.2 AF (Alignment-Free) approach:

Alignment-free approaches to sequence comparison can be defined as any method of quantifying sequence similarity/dissimilarity that does not use or produce alignment (assignment of residue–residue correspondence) at any step of algorithm application.(Zielezinski et al. 2017) Most of these methods are based on word statistics or word comparison, and their scalability allows them to be applied to much larger data sets than conventional Multi SA-based methods.

Alignment-free approaches can mainly be categorized in two groups:

word-based methods, methods based on the frequencies of subsequences of a defined length k (i.e. kmers) and **information-theory based methods**, methods that evaluate the informational content between full-length sequences.(Zielezinski et al. 2017) All the alignment-free approaches are mathematically well founded in the fields of linear algebra, information theory, and statistical mechanics, and calculate pairwise measures of dissimilarity or distance between sequences.

VI.ii.2 BoW method - Alignment-Free approach

Natural Language Processing (NLP) is the process of retrieving meaningful pieces of information from unstructured text data.

The Bag of Words (BoW) technique is a method used in NLP. It consists of converting the text data into words and their frequencies (counts per document) without giving importance to the order of the words.

A DNA sequence is a can be considered as an unstructured text data. This implies that NLP should be applicable on it.

In principle, similar sequences share similar kmers. We use distance measures on vector-represented kmers to evaluate the sequences' dissimilarity in terms of kmers occurrences.

Word frequency-based analysis approaches have been widely adopted in the bioinformatics community due to their efficiency and ability to accurately summarize and compare large datasets. Instead of performing inexact pattern matching, as aligners do, kmer based methods can simply examine the fraction of matching kmers within the query sequence(Marchet et al. 2020). The BoW is one of the two main approaches implemented in this research.

To implement the BoW model, we proceed by extracting kmers from the dataset to represent the features or the dimensions. The value of each dimension would be the frequency (number of occurrences) of this dimension (kmer) within a given sample (sequence). There are two main ways of kmer extraction:

1. **Overlapping kmers:**

As illustrated below in Figure 3, they are a sequence's subsequences of length k, such that they are generated using a window with a size k and step usually 1.

For example, the following table represents kmers of sizes from k=1 to k=7 generated from a short sequence GTAGAGCTGT.

k	k-mers
1	G, T, A, G, A, G, C, T, G, T
2	GT, TA, AG, GA, AG, GC, CT, TG, GT
4	GTAG, TAGA, AGAG, GAGC, AGCT, GCTG, CTGT

Table 1 k-mers for GTAGAGCTGT sequence

2. **Non-overlapping kmers (or n-grams):**

As illustrated below in Figure 3, they are a sequence's subsequences of length k, such that they are generated using a window with a size k and the step is also of size k.

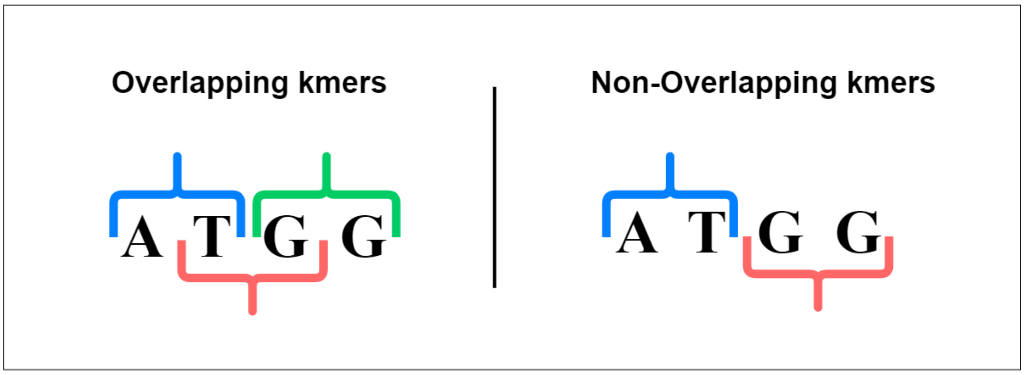


Figure 3 Overlapping vs non-overlapping kmers. In the first, a window of size k is moving with step n<k. In the second, the window of size k is moving with step of size n = k

VI.ii.3 Information theory-based Alignment-Free methods

The amount of information shared between sequences is computed as a measure of dissimilarity. We usually use compression algorithm to measure the percentage of compression of a sequence (algorithms like those used in zip). If the degree of similarity is high between two individual sequences, then the compression level of their concatenation is very close to their individual compression level. Whereas if they are less similar, their concatenation produces a compression level that is proportional to their cumulative compression levels.(Zielezinski et al. 2017). These methods are out of the scope of this study.

iii. Feature Selection Methods for NLP based classification

In highly dimensional data, the selection of subset of features is essential to reduce overfitting, reduce training time, improve the learning efficiency and to increases the predictive performance. Feature selection techniques reduce the learning set dimension by discarding irrelevant and/or redundant features.

There are two main types of feature selection techniques: supervised and unsupervised. Only supervised methods are relevant to this study.

Supervised methods may be divided into three main categories: wrapper, filter and intrinsic(embedded).

1. **Filter methods** use statistical measures to score the correlation or dependence between input variables that can be filtered to choose the most relevant features. (Jason Brownlee 2020)
   In this study, ANOVA (ANalysis Of Variance) correlation coefficient (linear), a univariate statistical measure, is used. It is adapted to problems of form "Numerical Input, Categorical Output". (Jason Brownlee 2020)
2. **Wrapper methods** considers the selection of a set of features as a search problem. It follows a greedy search approach by evaluating all the possible combinations of features against the evaluation criterion. The evaluation criterion is simply the performance measure, which for classification can be accuracy, precision, recall, f1-score, etc. A combination of features that gives the best results for the specified machine learning algorithm is selected.
   Most used techniques under wrapper methods are: Forward selection, Backward elimination and Bi-directional elimination(Stepwise Selection).
   In this study, Bi-directional elimination is used. Implemented using SequentialFeatureSelector of mlxtend python library.
3. **Embedded methods** complete the feature selection process within the construction of the machine learning algorithm itself. They consider each iteration of the model training process and carefully extracts those features which contribute the most to the training for a particular iteration.(Aman 2020). Examples are regularization and tree-based methods.
   In this study, tree-based method is used with Random Forest as algorithm. Random forests provide feature importance using mean decrease impurity and mean decrease accuracy methods. (Charfaoui 2020)

iv. LSTM based classification methods.

An interesting deep-learning architecture that deals well with sequential data inputs is the RNN (Recurrent Neural Networks). RNN however may suffer from the problem of vanishing and exploding gradients. This is where LSTM architecture (Long Short-Term Memory) would be a better architecture that deals with vanishing gradients problem. In this research, I am inspecting the possibility of the application of LSTM for the sequence classification with:

1. Raw DNA sequences without feature selection
2. Kmer encoded sequences.

v. LSTM Sequence Encoding methods

The genomics data needs to be transformed into numerical values (Matrix input) to be used in deep learning models. A good data representation is indispensable to get the best out of a model in terms of prediction quality.

There are three methods for sequence encoding: sequential encoding, one-hot encoding, and kmer encoding(Choong and Lee 2017).The characteristics of the three DNA encoding methods are shown in Table 2 Sequence Encoding Methods (Yang et al. 2020).

	Method	Description
1	Sequential Encoding	This method encodes each base as a number. For example, change [A,T,G,C] to [0.25, 0.5, 0.75, 1.0), and any other character can be encoded as zero.
2	One-hot Encoding	This method is widely used in deep learning methods. For example, [A,T,G,C] will become [0,0,0,1], [0,0,1,0], [0,1,0,0], [1,0,0,0]. These coded vectors can be connected or turned into a two-dimensional array.
3	Kmer Encoding	We take a longer biological sequence and decompose it into k-length overlapping fragments. For example, if we use a segment of length 6, "ATGCATGCA" will become: "ATGCAT," "TGCATG," "GCATGC," "CATGCA." This encoding needs another numerical encoding on top of it.

Table 2 Sequence Encoding Methods (Yang et al. 2020)

The performance of sequential encoding is known to be comparable to one-hot encoding, but the training time is significantly reduced. One-hot encoding is widely used in deep learning methods and is very suitable for algorithms such as CNN (convolutional neural networks). Performance of one-hot encoding is known to be good given that a suitable CNN is chosen.(Yang et al. 2020). Sequential encoding is used in this study.

vi. Sequence Embedding.

Embedding a sequence in a lower dimensional vector space makes the data more amenable for use by current machine learning tools, provided the quality of embedding is high and it captures the most meaningful information of the original sequences.(Kimothi et al. 2016)

The idea of using embeddings is to create a possibly explainable dissimilarity measure when applied to DNA sequences. A possible scenario is to use this measure on embeddings, for example with a KNN classifier, as it makes direct use of distances for the neighbor voting process.

Table 3 lists a selection of embedding methods recently published that are good examples.

Method	Description	Summary	Source	Code	Date
DNA2vec	Vector representations kmers	Based on Word2Vec, it converts variable length kmers into a n-dimensional vector bringing them in a Euclidean space.	(Ng 2017)	dna2vec Github	2017
Gene2vec	Representation based on co-expression	It utilizes transcriptome-wide gene co-expression to generate embedded gene vectors, a distributed representation of genes. This enables deep learning-based prediction tasks with gene names as input. These vectors capture functional relatedness of genes. Example task: gene-gene interaction prediction.	(Du et al. 2019)	Gene2vec Github	2019
Sec2vec	A method to embed an arbitrary biological. sequence into a vector space	Based upon BioVec, which is a word2vec based approach, but improved by using the doc2vec framework instead.	(Kimothi et al. 2016)	N/A	2016

Table 3 sequence Embedding Methods in recent research.

These methods have the advantage of allowing classification based on similarity. Sequence similarity means that there are similar or identical sites between sequences. If the degree of similarity between two sequences exceeds 30%, it is considered that the two sequences have a homologous relationship.(Yang et al. 2020) Many dissimilarity measures based on kmer analysis exist, including conventional (e.g., Euclidean, Manhattan, d2), and others based on presence/absence of kmers (e.g., Jaccard and Hamming distances), as well as state-of-the-art measures based on background adjusted kmer counts.

These methods may allow the use the similarity measure by various algorithms, such as KNN or deep learning.

vii. Model evaluation methods

To evaluate the performance of various algorithms used for classification, a set of benchmarks needs to be used. This includes accuracy, precision, f1-score, recall and AUC.

Accuracy (error rate) measures how often the algorithm classifies a data point correctly. Accuracy is the number of correctly predicted data points out of all the data points. More formally, it is defined as the number of true positives and true negatives divided by the number of true positives, true negatives, false positives, and false negatives. In certain problems, accuracy is not considered as an informative measure due to notion called Accuracy Paradox (Wikipedia contributors 2021a), and especially in the case of high class imbalance. However, in our problem, the classes are well balanced, so accuracy is an informative measure.

Precision is defined as the number of true positives as a proportion of all (true and false) positives, and it attempts to answer the following question: What proportion of positive identifications was actually correct?

Recall attempts to answer the following question: What proportion of actual positives was identified correctly?(Google n.d.)

F1 score conveys the balance between the precision and the recall. The samples are balanced across target classes hence the accuracy and the F1-score must be almost equal.

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

True Positive Rate

False Positive Rate

The formulae for these measures are found in the following Table 4 Formulae for prediction measures.

MEASURE	FORMULA
ACCURACY	$\frac{TP+TN}{Nn+Np}$
PRECISION	$\frac{TP}{TP+FP}$
SENSITIVITY	$\frac{TP}{Np}$
SPECIFICITY	$\frac{TN}{Nn}$
RECALL	$\frac{TP}{TP+FN}$
F1-score	$2*\left(\frac{PRECISION\ *RECALL}{PRECISION+RECALL}\right)$

Table 4 Formulae for prediction measures.

Where:

TP = number of true positives	TN = number of true negatives	Np: Full number of positives = TP + FN
FP = number of false positives	FN = number of false negatives	Nn: Full number of negatives = TN + FP

Implementation

VII. Study design

The experiment has three phases: preprocessing, training, and evaluation.

Each dataset is split into two subsets; a training set and a test set, where the training set is made up of 90% of the samples and the test set consists of 10%, given the number of samples available (60k). The splits are stratified for train, validation and test sets, which guarantees that each set contains approximately the same percentage of samples of each target class as the complete set. (Scikit-learn n.d.)

The models are trained using KFold cross validation. To adjust the class balances across training and validation, I use Stratified k-fold, a variation of k-fold which returns stratified folds.

Each machine learning algorithm is then fitted on the training data with n cross-validation, with n decided based on the time required to train. This step may be discarded if the training is very slow.

The performance metrics on the training data given by the cross validation is therefore the average of n different models. The trained algorithm is then applied to predict the test data on which test performance is evaluated.

The performance metrics of training and testing sets are then compared. AUC and accuracy are the main performance metrics used to compare the training and test sets as well as feature-selected and full datasets.

For the Bag of Words / NLP approach, and following the feature selection phase, the union of the best features is assembled, and used to train a model. Performances are compared to decide the best combination of features. The example below illustrates best features resulting from applying an embedded method.

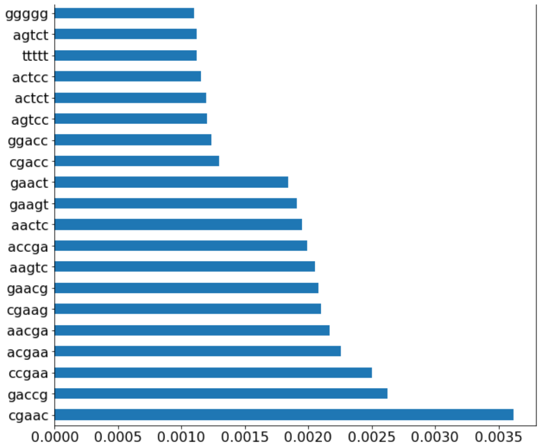


Figure 4 Best 20 features selected with an embedded Feature Selection method

For the LSTM approach, no feature selection takes place, and sequences are embedded into integers or one-hot vectors before being fed in for training.

For LSTM on kmer approach, sequences are kmer encoded, before being embedded into integers or one-hot vectors then are fed as input for training, then, hyperparameter tuning is applied to further optimize classifier.

Finally, the three approaches are compared based on the results.

VIII. Classification of Short Sequences

In the last chapter, I introduced methods found during my literature review, methods that are candidates to be implemented. I expose the three main implementation pipelines: BoW, LSTM and LSTM-on-Kmers.

i. Overview of the three Pipelines

In this chapter, I discuss my implementation. There are three pipelines as shown in Figure 5 below. Each pipeline consists of three main steps.

BoW Pipeline

1. Preprocessing:
   a. Clean, analyze and understand data.
   b. Extract kmers
   c. BoW Model: count kmers
   d. Feature selection
2. Exploring classifiers :
   a. Random Forest Classifier
   b. Logistic Regression
   c. SGD Classifier with hinge (similar to linear SVM)
   d. LSTM as a classifier (not a deep learning context).
3. Hyperparameters tuning.

LSTM Pipeline

1. Preprocessing:
   a. Sequence encoding into numerical values.
   b. Padding / truncating.
2. Exploring DL models :
   a. Simple LSTM layer
   b. Deeper mode
   c. Bidirectional LSTM
3. Training, hyperparameters tuning and evaluation.

LSTM on Kmer-encoded sequences Pipeline

1. Preprocessing:
   a. Extract kmers
   b. Sequence encoding into numerical values.
   c. Padding / truncating.
2. Exploring classifiers such as:
   a. Simple LSTM
   b. Deeper mode
   c. Bidirectional LSTM
3. Training, hyperparameters tuning and evaluation.

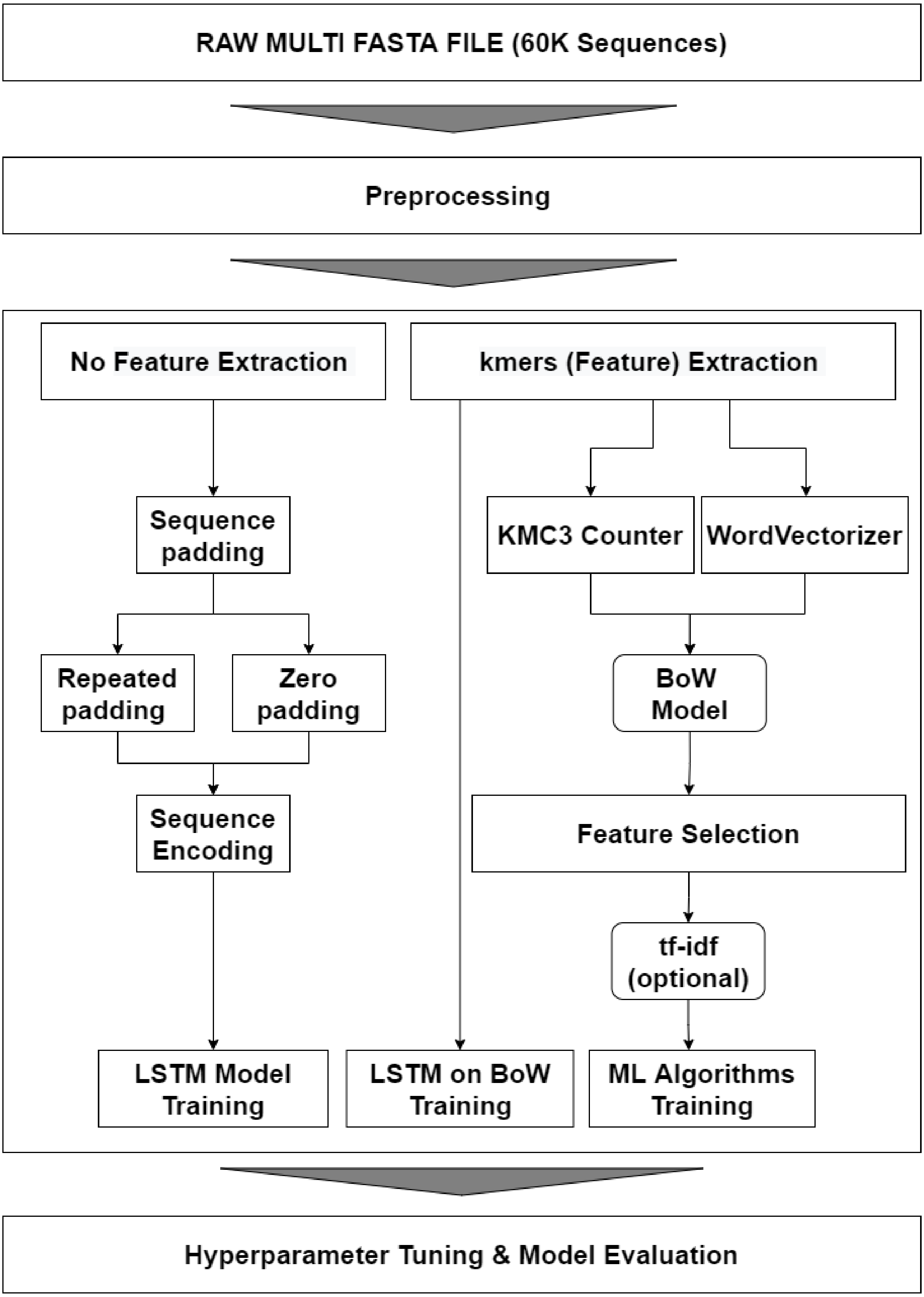


Figure 5 General Pipeline of the experiment

VIII.i.1 Data

VIII.i.1.1 Data Format

In bioinformatics and biochemistry, the FASTA format is a text-based format for representing either nucleotide sequences or amino acid (protein) sequences, in which nucleotides or amino acids are represented using single-letter codes.(Wikipedia contributors 2021c)

The raw data format is a multi-sequence file with FASTA format, of 600 megabytes size. The first line of a sequence starts with ">" followed by description data. The following lines contain the sequence data, composed of a permutation of four nucleotides (A,C,G and T), as the following example:

```
>SEQUENCE_1
CGTCGATCCAGACTTCGGAAAGCTGTCCAGCTGGATC
TAGCAGTAGACTAGCAGATACGATTGAGGGAGTCCG

>SEQUENCE_2
CTGTCCAGCTGGATCTAGCAGTACGTCGATCCGGACT
AGCATTGAGATACGAGAAAAACCTAGGGAGTCCGTC
```

Figure 6 An example of a multi-FASTA file

A multi-FASTA format, is simply a concatenation of multiple sequences in a single FASTA file, each starting with the description line as shown above in the example.

VIII.i.1.2 Data Preprocessing

Overview

At the first stage of the pipeline, the Data Preprocessing stage, data is transformed from the raw sequence format into a reusable data structure that can be handled by feature selection and ML techniques.

Along with Data Cleaning, the Data Preprocessing stage can be divided into two phases: Data Understanding and Data Preparation

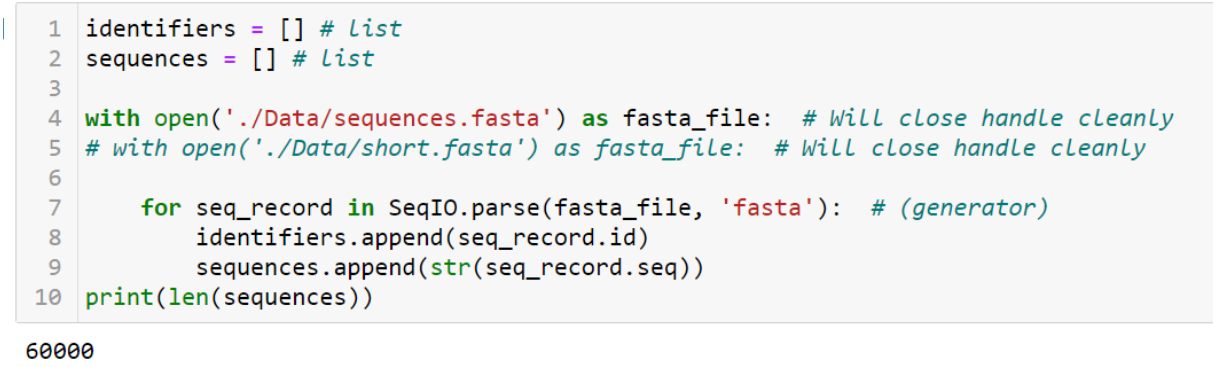
1. In the **Data Understanding phase** I do an Exploratory Data Analysis (EDA), where I describe, explore then verify Data quality and I clean it.
   To describe data, I compute basic statistics (e.g., distribution), which in this case can only be univariate with one attribute, the length of sequences. Then I explore data by analyzing the summary statistics obtained and try to extract meaningful insights.

2. In the **Data Preparation phase,** kmers are created and features based on these kmers are extracted. For the BoW (Bag of Words) approach, kmers were counted. This phase will be explained in detail later.

Data Cleaning

Import and clean raw fasta data.

At this stage, I transformed the multi-fasta file into multiple single Fasta file, each representing a DNA sequence using SeqIO.parse as shown below:



```
identifiers = [] # list
sequences = [] # list

with open('./Data/sequences.fasta') as fasta_file:  # Will close handle cleanly
# with open('./Data/short.fasta') as fasta_file:  # Will close handle cleanly

    for seq_record in SeqIO.parse(fasta_file, 'fasta'):  # (generator)
        identifiers.append(seq_record.id)
        sequences.append(str(seq_record.seq))
print(len(sequences))
```

```
60000
```

Figure 7 Shows how I split a multi-fasta file into multiple single fasta files.

After importing all the samples into one single Data Frame of sequences, I controlled the data for duplicate samples and noise, i.e., values other than ACGT, where I validate that all sequences are DNA, i.e., only contains four base nucleotides {A,C,G,T}.

Data Understanding:

Describe, explore then verify Data Quality - Exploratory Data Analysis (EDA)

I performed this step to understand the trends and patterns, to be able to analyze the frequency and other such characteristics, to know the distribution of the variables in the data and visualize the relationship that may exist between different variables.

Data Distribution

First, I plotted the labels to see the whether the data is balanced as shown in Figure 8 below. It shows that the data is balanced.

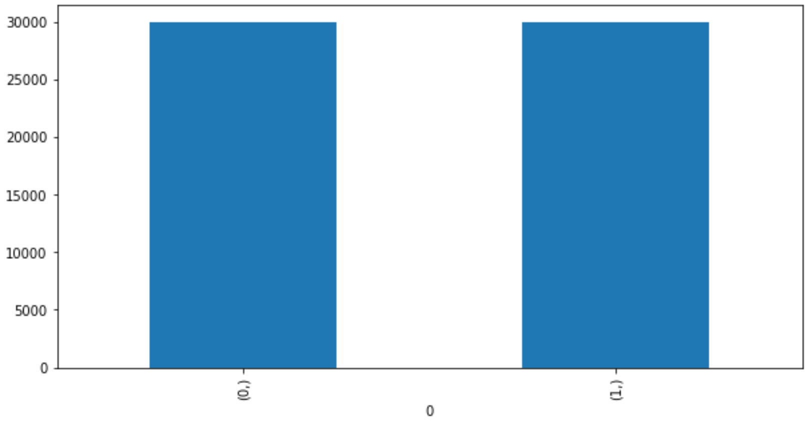


Figure 8 Plot showing data distribution over the two classes.

I then plotted a histogram, Figure 9, which is a representation of the distribution of the sequence's length variable, along with a density plot, a smoother version of a histogram. Highest density resides in the range of [~6k,~14k], leaving around ~2k sequences below average density.

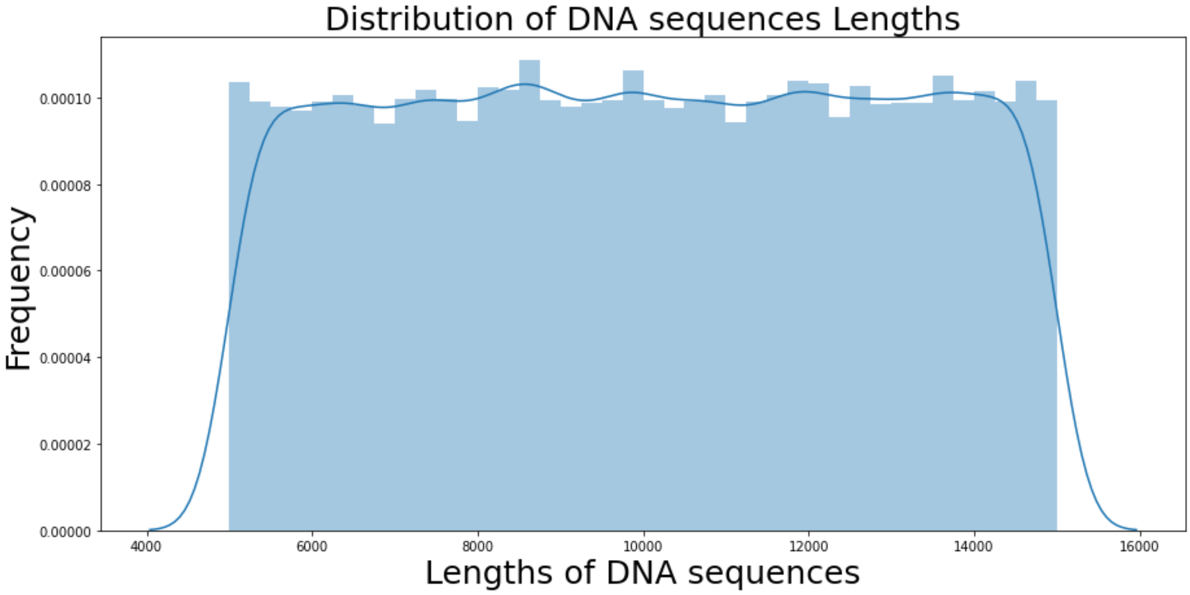


Figure 9 Histogram of sequences frequency

Normality test:

Finally, I plotted an ECDF plot along with a boxplot (Figure 10 below) to test the normal distribution of the sequences upon length variable. Based on literature (Sivasai Yadav Mudugandla 2020), these two plots show that the data may have a normal distribution over sequence length. The plots below display a uniform pattern in our artificially generated sequence data that is not found for real data. This uniformly increasing length is probably due to the sequence generation algorithm.

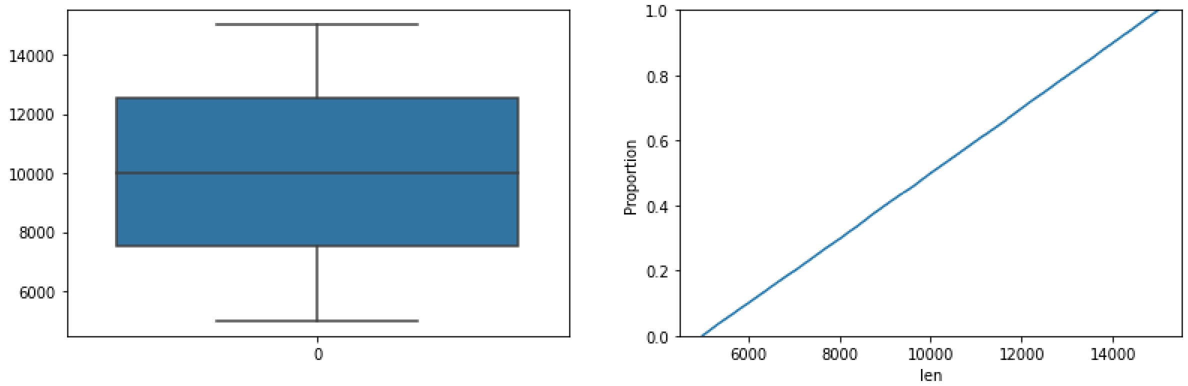
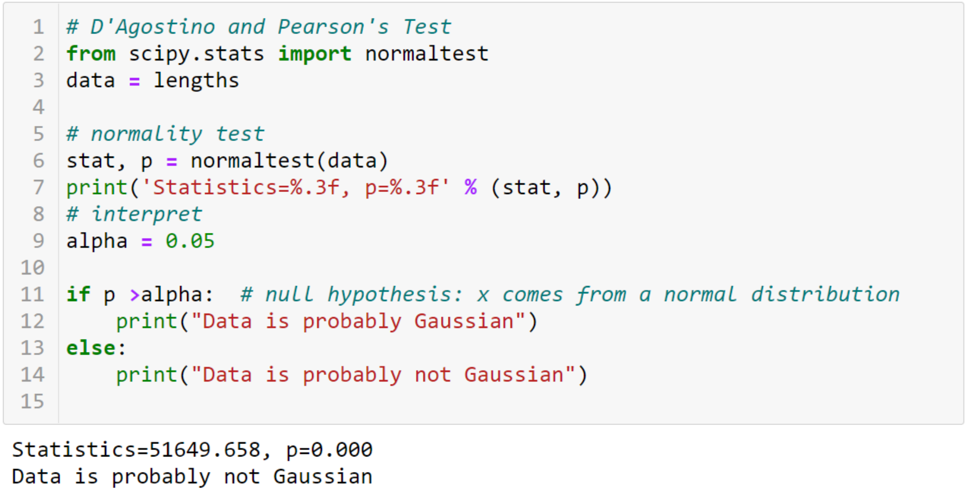


Figure 10 Box plot (left) ECDF plot (right)

However, to be sure of the hypothesis, I performed a statistical normality test called D'Agostino and Pearson's Test (Jason Brownlee 2018) and it returned a negative result (Figure 11), suggesting that the distribution is probably not normal. This is particularly due to the kurtosis property of the data.



```python
# D'Agostino and Pearson's Test
from scipy.stats import normaltest
data = lengths

# normality test
stat, p = normaltest(data)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05

if p >alpha:  # null hypothesis: x comes from a normal distribution
    print("Data is probably Gaussian")
else:
    print("Data is probably not Gaussian")

```

```
Statistics=51649.658, p=0.000
Data is probably not Gaussian
```

Figure 11 Normality test

NLP based Classification with

Bag of Words

model

ii. Classification with Bag of Words

VIII.ii.1 Feature Extraction (BoW)

Transforming FASTA files into a Data Frame with kmers as features and counts as values

The features are the count of each kmer in each sample. For a given kmer length k, I extract all the kmers within a sequence then count them. These are two major steps that I approach by finding the compromise between computing capacity in hands and time tolerance. In the following I explain how I implemented various methods starting from the most basic up to the most efficient and fastest method.

I experimented on at least 6 different implementations of the transformation step with minor versions in between, in an objective of increasing the performance, decreasing the required time and allow to scale up for higher values of k. The performances of each of the implementations is summarized in the Table 5 represents the performances of each implementation of the kmer counting. below.

Implementation	Technology used	Details	kmer	Time (hrs.)	Drawbacks
1	Pandas	Panda's map + apply Fails on k>=6	6	-	Memory Error for k>=6
2	Pandas	Parallel	6	**24**	Slow
3	Pandas, Dictionary multiprocessing	Parallel + Dictionary	6	5.7	Slow
4	KMC, Pandas, multiprocessing	KMC + Parallel concat	6	2	Slow
5	Pandas, CountVectorizer	CountVectorizer	6	**0.38**	limited to k=7

Table 5 represents the performances of each implementation of the kmer counting.

Feature extraction implementations:

First Method. A sequential implementation with Pandas Data-Frames where I create kmers from sequences then I count them and insert them into a Data Frame. This approach was a bit slow but very memory intensive and would crash with kmer sizes higher than four.
As a first attempt to creating the features (kmers), in first method I created a combination of all possible kmers of certain length k to create the header of an empty Data Frame using Pandas Data-Frames as they are reputed to be very fast and reliable. This Data Frame would contain all the extracted kmers starting with k=4, however with increasing size of kmers, the size of the data-frame would explode I would run out of memory with errors, leading to a conclusion that it is not a practical approach for kmer counting.

Second Method. A parallelized version of the first implementation. To bypass the memory problem, I wanted to parallelize using Pandas known to be memory tolerant. Parallelizing the code would solve the memory problem in the second implementation, but on the cost of time as it took around ~24 hrs. for kmers of size 6.

Third Method. An implementation that creates kmers and their counts with only one pass using dictionaries as data structures. Along with multiprocessing.
Wanting to implement a clean and faster, in the third implementation, I imagined using a different approach, this time using a different data structure, a dictionary, known especially for their speed. Also

using a dictionary would cut the number of passes by half after generating all the kmers, and for each kmer as a key, I increment the value (occurrence) by one, with only one pass over the dataset resulting with kmers with their frequencies in only one pass. However, the performance was not promising, ~5.7 hrs. for k=6.

Fourth Method. An implementation using KMC3, a state-of-the-art lightning-fast kmer counter written in C++ and a parallelized version of the third implementation.

Since KMC3 does not have the option to output counts per sequence (only per FASTA file, would it be single or multi), I had to split multi fast into many single-sequence files, apply KMC3 then reassemble the data into a single data structure. I created a python script notebook to automate this process. This method was a success with remarkable improvement from the previous ones, with a quasi linear time complexity versus the size of K (time taken for 4 counting is slightly less than the time taken for 8 counting).

But using this method is not practical for two reason. The first is that certain required python libraries were incompatible with ipython notebooks, and the second is the time it takes, believing I can do better ~2 hrs.

To resume the process, the idea is to:

1. Split a multi-fasta file into many single-fasta files
2. Then apply the KMC kmer counter
3. Then using a python script, read the output files into a dataframe each
4. Then concatenate into a single dataframe in parallel and write the final df to disk.

Command to to create a 10-kmer BoW with KMC:

```
kmer_counter.exe -k10 -fm sequences.fasta kmers10 C:\Users\KMC\3\sequences
```

To transform result into text file:

```
kmc_dump.exe shortkmers10 shortkmers10.txt
```

Fifth Method. The most efficient and fastest method to extract features was using Word Vectorizer.

This last attempt was the most successful where I learned more about Word Vectorizer and figured out a way to reformat the sequences so that they are compatible with it.

Custom preprocessing necessary for the Word Vectorizer to work, as it required that the list of kmers be under the form of one string with a separator (i.e., comma, space, etc.). Figure 12 below explains visually the input and output format of each Word Vectorizer preprocessing step.

This pipeline takes place in three consecutive steps where sequences undergo two different transformations before they can be handled by WordVectorizer. Due to the memory limitation on the experiment workstation (<8GB available memory), these three steps are implemented in three different ipynb notebooks as below, where the memory needed to be freed after each step.

Step 1 CV Pre-processing .ipynb

▼

Step 2 CV Pre-processing .ipynb

▼

Step 3 CV Pre-processing .ipynb

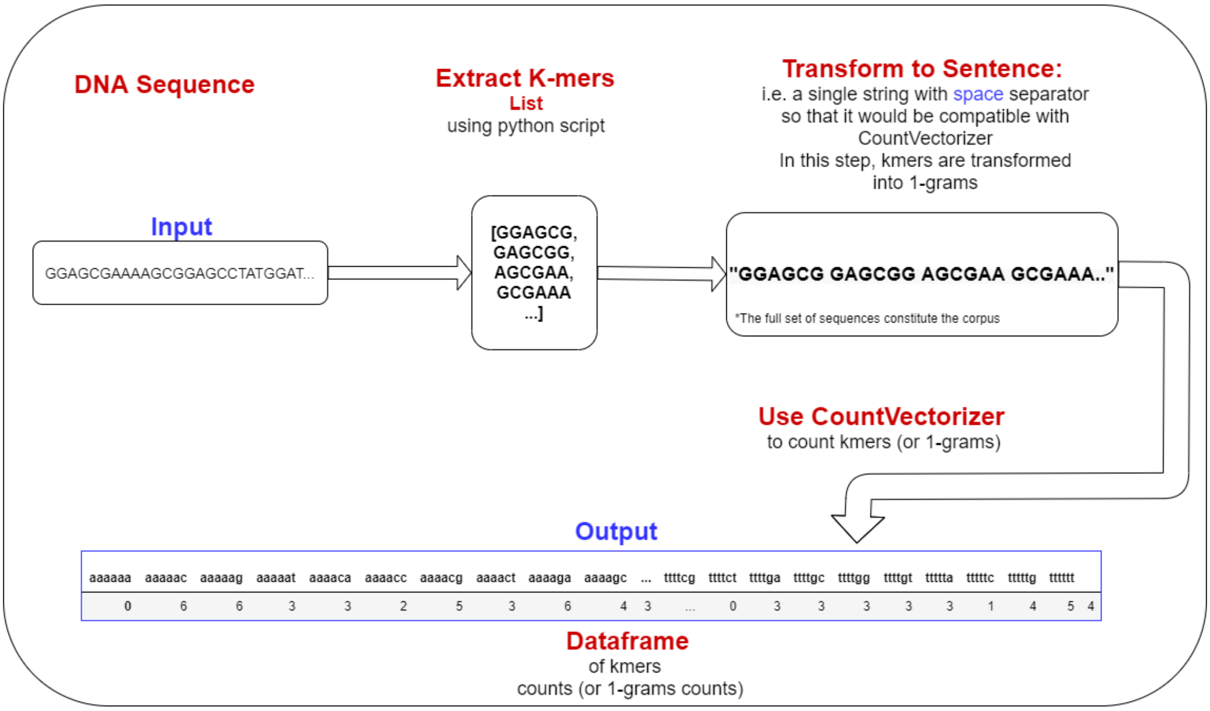

DNA Sequence
Extract K-mers
List
using python script
Transform to Sentence:
i.e. a single string with space separator
so that it would be compatible with
CountVectorizer
In this step, kmers are transformed
into 1-grams
Input
GGAGCGAAAAGCGGAGCCTATGGAT...
[GGAGCG,
GAGCGG,
AGCGAA,
GCGAAA
...]
"GGAGCG GAGCGG AGCGAA GCGAAA.."
*The full set of sequences constitute the corpus
Use CountVectorizer
to count kmers (or 1-grams)
Output
aaaaaa aaaaac aaaaag aaaaat aaaaca aaaacc aaaacg aaaact aaaaga aaaagc ... ttttcg ttttct ttttga ttttgc ttttgg ttttgt tttta tttttc tttttg tttttt
0 6 6 3 3 2 5 3 6 4 3 ... 0 3 3 3 3 3 1 4 5 4
Dataframe
of kmers
counts (or 1-grams counts)

Figure 12 Feature extraction pipeline using Count Vectorizer .

An interesting observation was that the kmer count becomes very flat with increasing K>6, thus resulting in a highly sparse matrix with exploding feature space, where most of the counts being rare (counts under a certain threshold if set). Removing rare kmers would result in a much smaller feature space. Although later in processing I would need to scale up for full feature space (4^K) to have a uniform dataset, thus resulting in an even more sparse matrix, but this time, I would apply feature selection and/or dimensionality reduction, thus only keeping the important and informative features.

This method improved the speed (runtime) with a factor of 63 (0.38 hrs), in comparison with the slowest method 2 (24 hrs.) on the full dataset of 60K sequences, with sequence length ranging between 4,999 and 14,999 characters.

VIII.ii.2 Feature Selection, Tuning, and evaluation

I designed two feature selection pipelines. One is purely based on embedded method, a selection capability within the Random Forest Classifier. And the second is based on statistical measures independent of the classifier.

Two designs were developed for two reasons:

1. *To include more than one selection method (embedded and filter) as a good rule of thumb.*
2. *To prevent potential bias of using the same algorithm for feature selection and classification in the case of first selection design.*

Due to hardware limitations, the maximum size of kmer studied in this research is k=7, thus 7 datasets were generated each corresponding to a kmer length, ranging between k=1 and k=7

Given the set of nucleobases {A,C,G,T} of cardinality C=4, the maximum number of features a feature space can have equals to C^K.

For K=5, 6, 7, the number of features can reach 1024, 4096, 16384 respectively, which results in a high dimensional feature space. Hence, we need to apply feature selection, which is the process of selecting a subset of relevant features that have the most effect on classification quality.

For K=1, 2, 3 and 4, no feature selection is necessary since they have a low dimensional space.

VIII.ii.2.1 Effect of varying the length of kmer on performance.

The objective of this analysis is to decide which kmers to include in the feature selection process. Thus, I study the classification power of each dataset with k=1 to k=7.

I trained and evaluated three different classifiers:

1. Random Forest Classifier: although it offers less interpretability of classification, it is known to perform very well on high dimensional data, it is parallelizable, and it is robust to outliers and non-linear data. It however can tend to overfit, so we need to tune the hyperparameters later. (Julia Kho n.d.)
2. Logistic Regression
3. SGD Classifier: when trained with the hinge loss, is equivalent to a linear SVM.(Scikit-learn n.d.)

The results presented in Figure 13, Figure 14 and Figure 15 below suggest that increasing the size of k results in better performance. By increasing the number of dimensions, more information is extracted, enabling classifier to differentiate classes. We also notice that the basic dimensions with k=1, k=2, k=3 and k=4, when used isolated, do not hold enough information to enable correct classification.

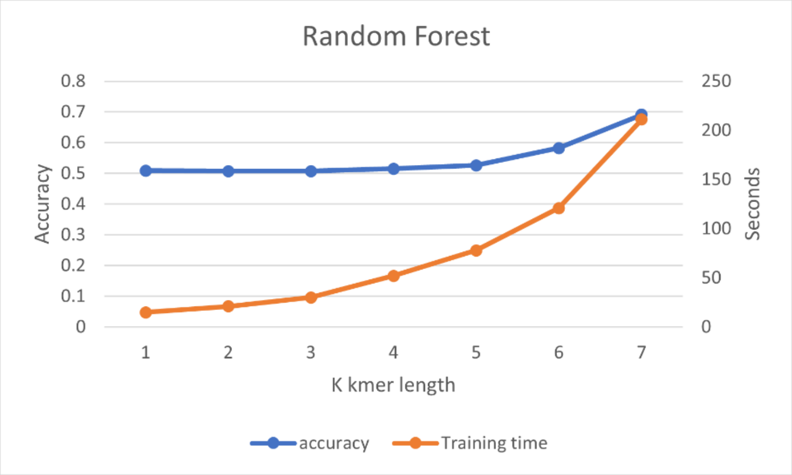


Figure 13 Effect of kmer length on Random Forest accuracy

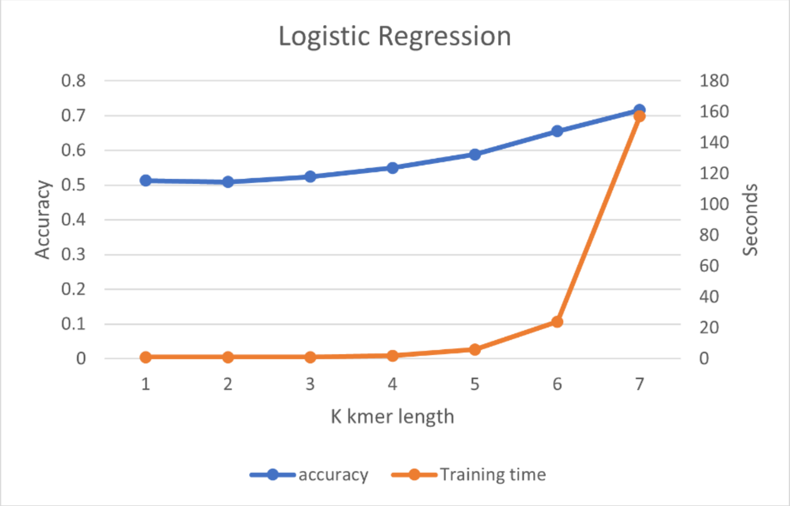


Figure 14 Effect of kmer length on Logistic Regression accuracy

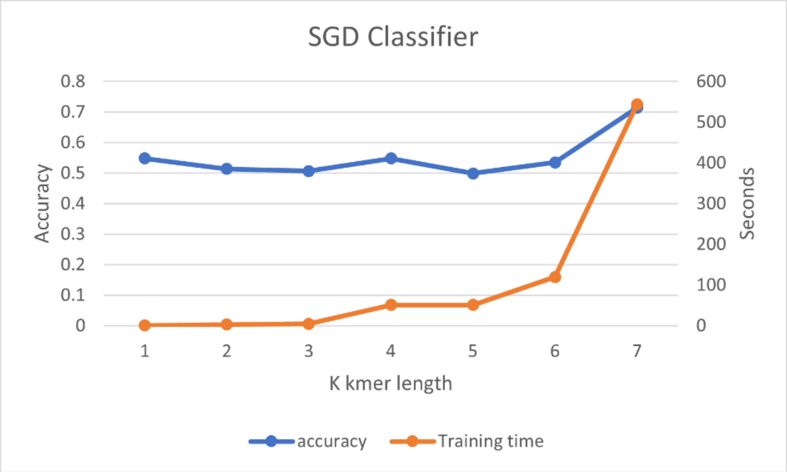


Figure 15 Effect of kmer length on SGD Classifier's accuracy

Detailed statistics on this test can be found in the annex, presented with confusion matrices and Table 16 to Table 22.

Analysis of effect the length of kmer on performance:

The increase of performance for Random Forest Classifier between k=1 and k=5 is of 3%, which is relatively low.

The increase of performance for Random Forest Classifier between k=5 and k=6 is of 11%, which is relatively moderate.

The increase of performance for Random Forest Classifier is particularly most dramatic between k=6 and k=7 with an average increase of 19% .

We notice that the data exhibits a higher discriminative potential with higher kmer lengths (more dimensions), thus the performance of the Reference classifiers increases with the increase of k (kmer length).

It is clear that 7-mers have the highest effect on performance score and are the most promising for the classification problem in hand.

6-mers come in second place in terms of effect on performance, followed by 5-mers.

Based on results, and to avoid exploding feature space, I decided to nominate datasets with kmers of length k=5, k=6 and k=7 and discard the others k = 1, 2, 3, 4.

After selecting the best kmer length, I next apply feature selection design 1 and 2 and compare them in terms of the effect on classification performance.

VIII.ii.2.1 Feature Selection Design 1 and Results: Embedded + Feature number Search methods

The general approach goes as follows:

Random Forest Classifier has an embedded method that selects best features. I use a wrapper method that I designed, to search for the best number of features among selected ones. I consider it as a wrapper method because it takes into consideration the direct effect of number of features on the classification performance.

1. For each of the three datasets (k=5, 6, 7)
   - Apply feature selection using Random Forest's embedded method.
   - Discover the best number of features to select using the wrapper-search method.
   - Train three classifiers on each dataset and evaluate performance.
   - Datasets with performance >= 70% are nominated for the next stage
   - Combine selected features from each nominated dataset.
2. For the combined dataset:
   - Apply a second round of feature selection using Random Forest's embedded method.
   - Discover the best number of features to select using the wrapper-search method.
   - Train three classifiers and evaluate performance.
3. Hyperparameter Tuning and evaluation.

Design 1 – Stage 1: Decide which datasets to combine out of k=5, k=6 and k=7.

At this stage, the impact of feature selection on each dataset is measured to nominate the best performing datasets given a threshold of performance of 70% (chosen arbitrarily as a rule of thumb). Thus, a dataset is nominated for next stage if the AUC score is greater than 70%.

For each datasets of k=5, k=6 and k=7, best features are selected using this embedded method, and as shown in Figure 16 Figure 19 and Figure 22 below, best features are ranked according to importance.

Due to their relatively low impact on AUC score with Random Forest, features with k=5 are discarded and are not considered for the next stage of feature selection on combined datasets.

Feature Selection for k=5

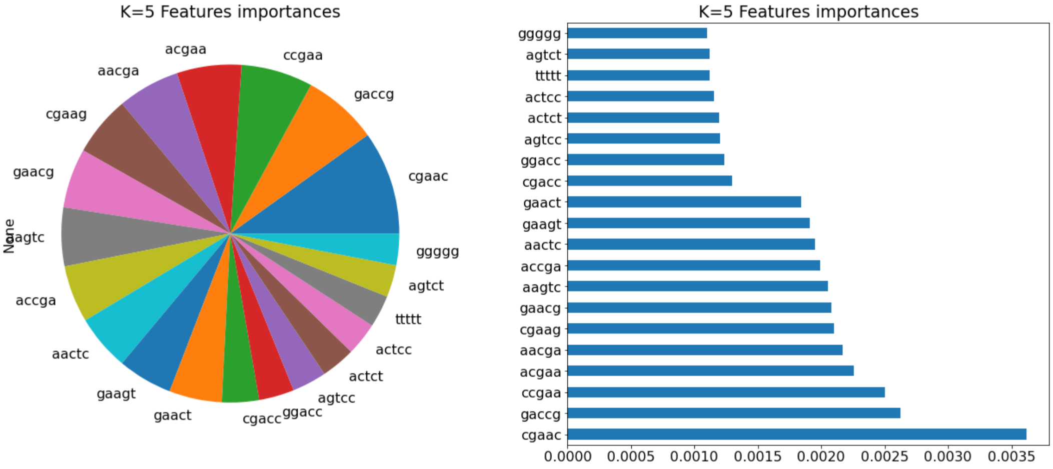


Figure 16 Feature importance score from Random Forest Classifier k=5. Left: pie chard. Right: Bar diagram. Both diagrams present the same information in two visually different ways, allowing for the reader's preference.

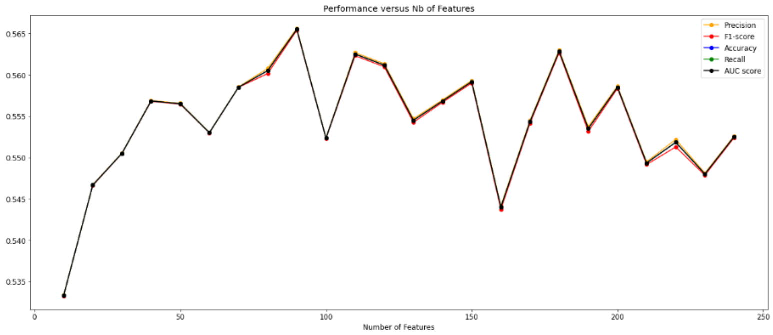


Figure 17 This graph shows performance on k=5 while searching for the best number of features, starting with the most important features, adding one feature at a time and repeating training.

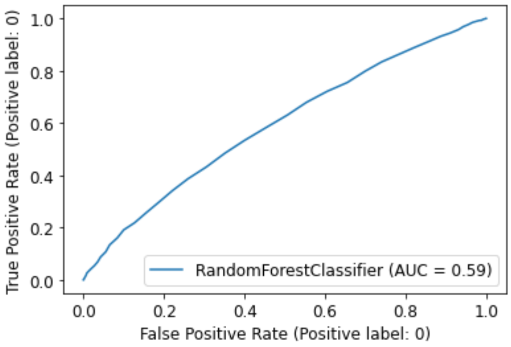


Figure 18 AUC Score for Random Forest Classifier, k=5

Classifier	accuracy	precision	recall	f1-score	AUC Score	Features
RandomForestClassifier	0.564	0.564	0.564	0.564	0.587	70
LogisticRegression	0.583	0.583	0.583	0.583	**0.615**	70
SGDClassifier	0. 717	0. 717	0. 717	0. 717	N/A	70

Table 6 Test performances with k=5, of three different classifiers with best features based on Random Forest Classifier. The last column represents the number of features that scored highest.

Feature Selection for k=6

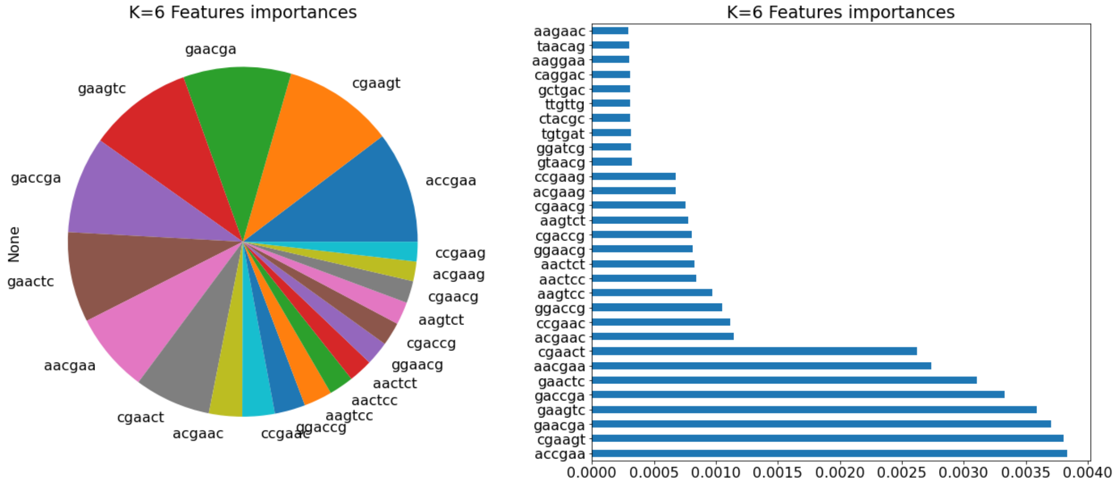


Figure 19 Feature importance score from Random Forest Classifier k=6

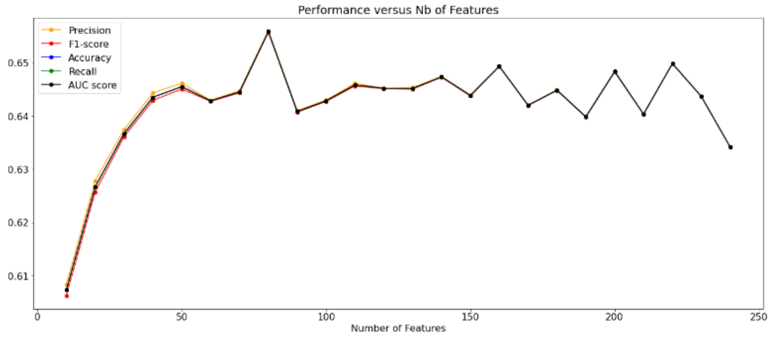


Figure 20 This graph shows performance on k=6 while searching for the best number of features, starting with the most important features, adding one feature at a time and repeating training.

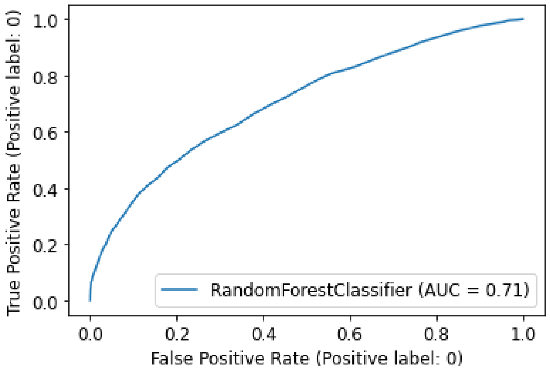


Figure 21 AUC Score for Random Forest Classifier, k=6

Classifier	accuracy	precision	recall	f1-score	AUC Score	Features
RandomForestClassifier	0.656	0.656	0.656	0.656	**0.709**	80
LogisticRegression	0.65	0.65	0.65	0.65	0.709	80
SGDClassifier	0.652	0.652	0.652	0.652	N/A	80

Table 7 Test performances with k=6, of three different classifiers with best features based on Random Forest Classifier. The las column represents the number of features that scored highest.

Feature Selection for k=7

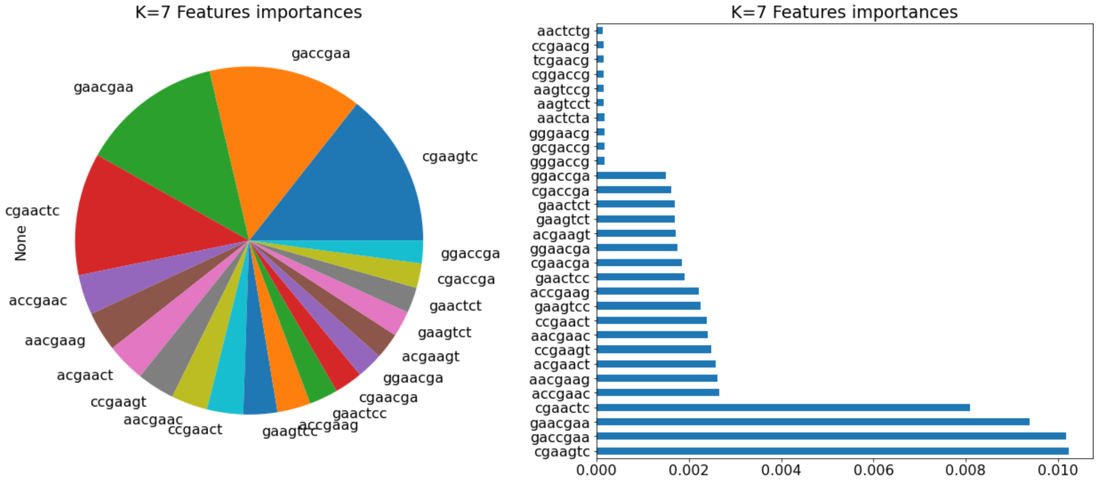


Figure 22 Feature importance score from Random Forest Classifier k=7

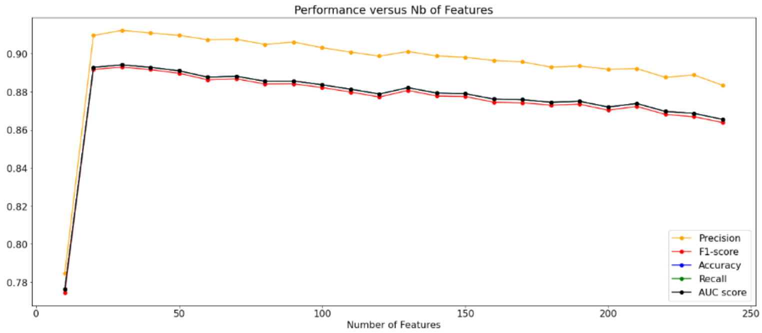


Figure 23 This graph shows performance on k=7 while searching for the best number of features, starting with the most important features, adding one feature at a time then repeating training.

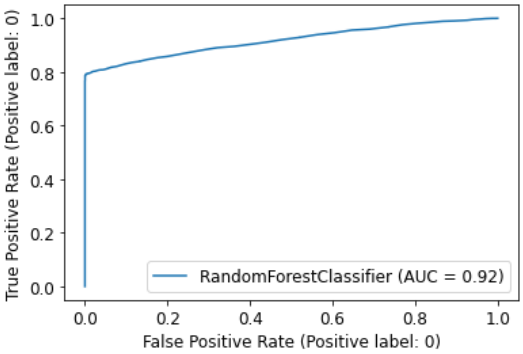


Figure 24 AUC Score for Random Forest Classifier, k=6 + k=7

Classifier	accuracy	precision	recall	f1-score	AUC Score	Features
RandomForestClassifier	0.894	0.912	0.894	0.893	**0.918**	25
LogisticRegression	0.713	0.713	0.713	0.713	**81.4**	25
SGDClassifier	0. 732	0. 732	0. 732	0. 732	N/A	25

Table 8 Test performances with k=7, of three different classifiers with best features based on Random Forest Classifier. The las column represents the number of features that scored highest.

Design 1 – Stage 2: Embedded method on combined dataset k=6 and k=7.

At this stage, best features from k=6 and k=7 are combined in a single dataset while discarding features with k=5. In a hope to obtain better performance, a second round of feature selection using Random Forest is applied.

Features scores are shown in Figure 16, Figure 19 and Figure 22.

This is followed by a search to decide on the best number of features to select as shown in Figure 25 and Figure 26 on page 41.

Then three algorithms are benchmarked with the new feature set, and results are shown in Table 9.

In the last stage, hyperparameter tuning is applied to further improve the model.

Feature Selection for combined dataset from k=6 + k=7

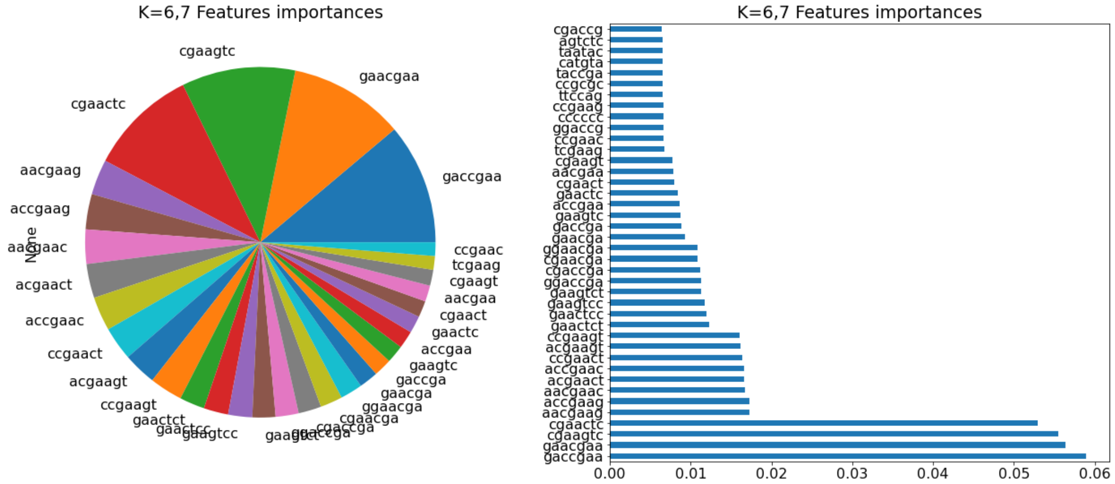


Figure 25 Feature importance from Random Forest Classifier k=6+7

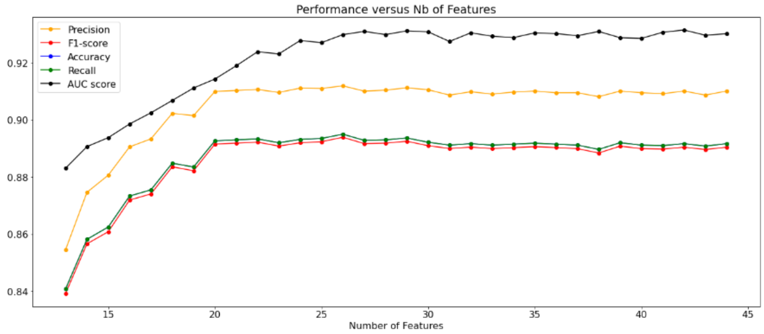


Figure 26 Graph showing the performances with k=6 + k=7, of Random Forest Classifier w.r.t. total number of features. This search is used as a heuristic wrapper method to select the best number of features

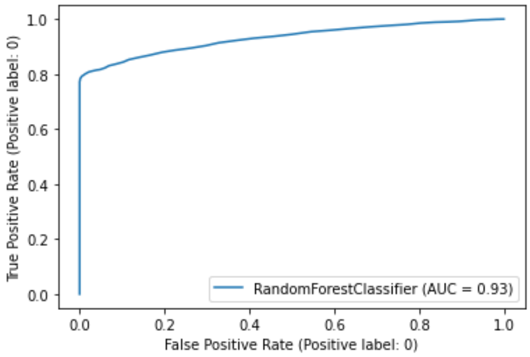


Figure 27 AUC Score for RFC, k=6 + k=7

Classifier	accuracy	precision	recall	f1-score	AUC Score	Features
RandomForestClassifier	0.892	0.910	0.892	0.890	**0.931**	42
LogisticRegression	0.798	0.798	0.798	0.798	0.871	42
SGDClassifier	0.797	0.798	0.797	0.797	N/A	42

Table 9 Test performances of three classification algorithms on combined dataset k=6 and k=7.

Design 1- Stage 3 : Hyperparameters Tuning

At this stage, we use the selected features from stage 2 to perform hyperparameter tuning. Tuning was performed on RandomForestClassifier, LogisticRegression and SGDClassifier using RandomizedSearchCV to minimize the tuning time, and results obtained are presented in Table 10 below.

We notice that hyperparameter tuning did not drastically enhance the performance in comparison to results obtained from stage 2.

Classifier	accuracy	precision	recall	f1-score	AUC Score	Features
RandomForestClassifier	0.889	0.909	0.889	0.887	**0.934**	42
LogisticRegression	0.798	0.798	0.798	0.798	0.871	42
SGDClassifier	0.779	0.779	0.779	0.779	N/A	42

Table 10 Test performances of three different classifiers after Hyperparameter Tuning of FS Design 1

To study the effect of hyperparameter on training and test scores, sklearn.model_selection.validation_curve is used as shown in Figure 28 below, which plots the variation of score with respect to a given hyperparameter. The models have shown to be non-sensitive to certain hyperparameters as train score did not vary significantly while validation score staying stable.

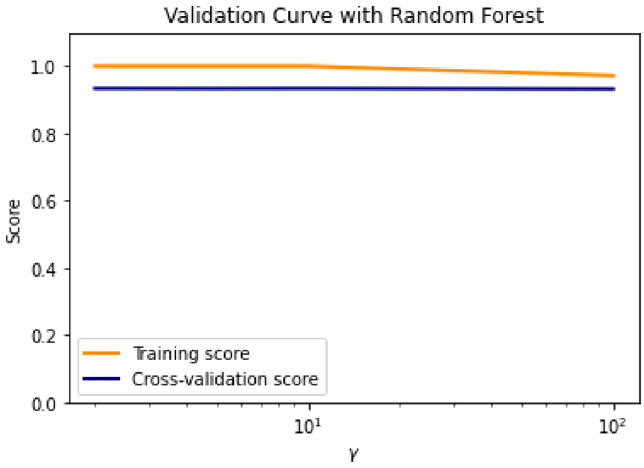


Figure 28 Validation curve of Random Forest model with respect to some hyperparameter

Noting that using the same algorithm (Random Forest) for both feature selection and prediction may result in a biased classification, I created a second design of feature selection, applying filter-based selection method first, then applying embedded method of Random Forest.

VIII.ii.2.2 Feature Selection Design 2 and Results: Filter + Embedded + feature number Search methods

To avoid possible bias introduced by using the same algorithm for feature selection and classification, I added an extra selection that uses statistical-based feature selection algorithms, namely, Fischer score and Gini Score.

The general approach goes as follows:

1. Filter methods: For each of the three datasets (k=5, 6, 7)
   - Apply filter feature selection based on Fischer score.
   - Apply filter feature selection based on Gini score.
   - Search for best number of features on the results of each of the previous methods using the wrapper search method.
   - Create a union of the best performing features.
   - Combine selected features from each selected dataset. (k=5, 6, 7)
2. Embedded method: on the features obtained from previous stage.
   - Apply a second round of Feature Selection using embedded method.
   - Search for best number of features using the wrapper search method.
   - Train three classifiers on each and evaluate performance.
3. Hyperparameter Tuning and evaluation.

Design 2 - Stage 1 : Using Filter methods

First, I select best features from each dataset of sizes k=5, k=6 and k=7 with two different methods, Gini-Score and Fischer-Score, both being filter methods.

I then search for the number of best performing features using the search method, then I create a union of features.

Best number of features is extracted by an incremental sequential search using Random Forest Classfier, where the number of features is incremented at each iteration and performances are plotted as shown in Figure 29 below.

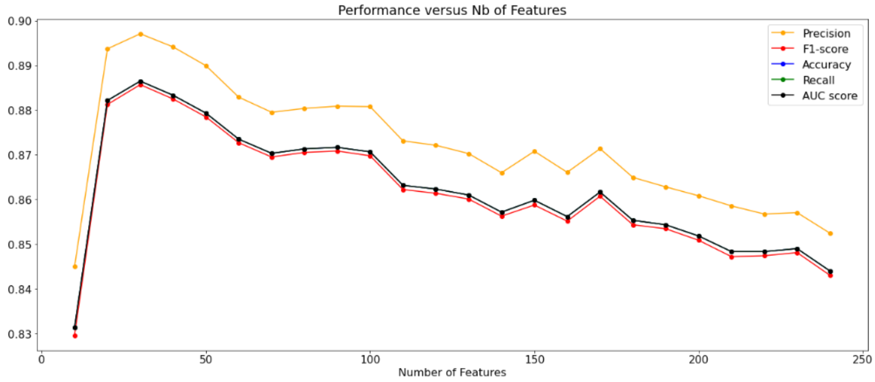


Figure 29 Performance of RFClassifier versus number of features. This is a sequential feature space search on a given range.

The union of the best features from filter methods is created and passed for the next selection method.

The most important performance measure used in deciding on the best number to select is the ROC-AUC.

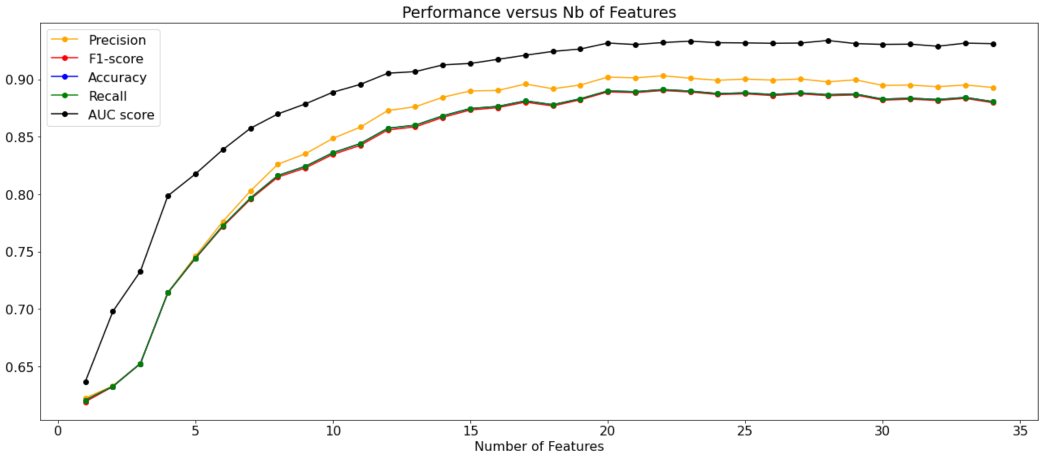
Once the best number of features is extracted at this stage, the performances of three different classifiers are benchmarked to evaluate the improvement of performance.

Classifier	accuracy	precision	recall	f1-score	AUC Score	Features
RandomForestClassifier	0.889	0.88	0.889	0.88	**0.922**	27
LogisticRegression	0.755	0.755	0.755	0.755	0.85	27
SGDClassifier	0.759	0.76	0.759	0.759	N/A	27

Table 11 Test performances of three different classifiers after stage 2 of FS design2.

Design 2 - Stage 2 : Using Embedded method

Another round of Feature selection is applied using Random Forest Classifier embedded method. Then a sequential search is applied again to take the best number of features.



Design 2 - Stage 3 : Hyperparameters Tuning Design 2

Finally, hyperparameter tuning is applied to conclude which is the best FS design. To minimize the training time, RandomizedSearchCV is used, and results obtained are presented in Table 12 below.

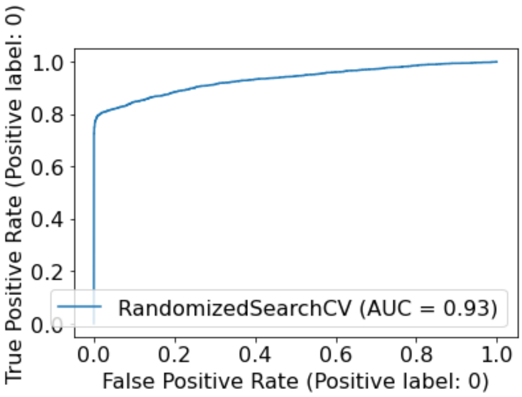


Figure 30 AUC Score for RFC after tuning

Classifier	accuracy	precision	recall	f1-score	AUC Score	Features
RandomForestClassifier	0.889	0.903	0.889	0.888	**0.933**	27
LogisticRegression	0.782	0. 782	0. 782	0. 782	0.864	27
SGDClassifier	0.779	0.779	0.779	0.779	N/A	27

Table 12 Test performances of three different classifiers after Hyperparameter Tuning of FS Design 2

Analysis of results

As a reminder, I developed the second selection design for two reasons:

1. *To prevent potential bias of using the same algorithm for selection and classification in the case of first selection design.*
2. *To use more than one selection method (embedded and filter) as it is considered a good practice.*

Both FS designs have shown almost similar results in terms of performances on test set. Each stage of the design 1 and 2 has demonstrated improvement of the overall classification performance.

Although they have the best scores, the first 20 features of embedded method (in design 1) do not necessarily provide the best performance (as can be seen in figures Figure 17 and Figure 20) except for k=7 (Figure 23). This may imply that features with a score lower than the best twenty, can still have potential to increase the discriminative power of classifiers.

An interesting observation is that most important features (kmers) have common patterns. From design 2, common patterns appearing in the first 24 are "cgaa", "gaagt", "accga", "aacga" and "gaact" as shown in Figure 31 below.

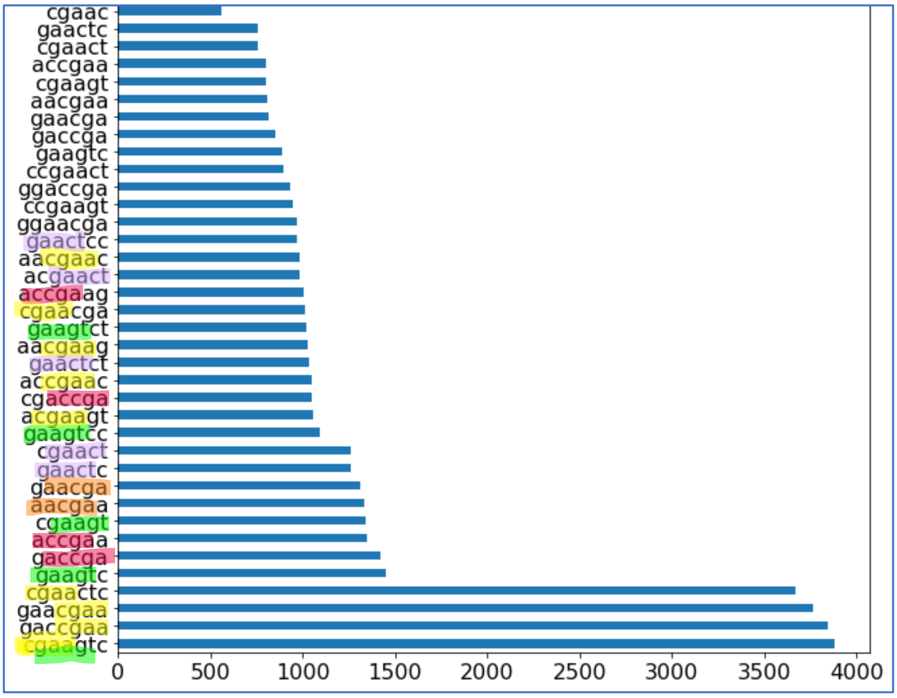


Figure 31 Common patterns in the most important features (kmers) resulting from filter methods of design 2.

Similar patterns can be observed on best features using the embedded feature selection.

While both filter and embedded methods give more importance to 7-mers, followed by 6mers, filter methods rank 6mers higher than the embedded method does. This can be shown by counting the occurrence of each 7mers and 6mers among the first 25 most important features, as shown in Table 13 below.

	Rate of 6-mers	Rate of 7-mers
Filter Method	8/25	17/25
Embedded Method	5/25	20/25

Table 13 The rate of 7-mers and 6-mers in the first 25 most important features by selection method

The performance of RandomForest on training set is 99% for both designs. This indicates that - if this is to be considered overfitting - it is probably not due to the bias introduced by design 1 (using same algorithm for selection and classification), but rather due to the ability of the model to catch on details. Yet the difference in performance between training and testing sets (0.99 vs 0.94 AUC) can relatively be considered as over-fitting. This may be mitigated by training on more data, or possibly using deep learning.

The other two classifiers, Logistic Regression and SVM (SGD Classifier with hinge) are simpler than RandomForest, thus it is not surprising that they did not perform that well on the same features. Extracting longer kmers (k>=7) will probably allow for better performance with the simpler classifiers, and will also allow the use of RandomForest classifier with fewer number of trees, but will also mean that we have to deal with much higher number of features.

VIII.ii.2.2.1 Using LSTM as classifier on selected features:

To get an idea of how a LSTM classifier would perform as a classifier on the selected features, I trained a six-layers LSTM model with 379,859 parameters (Figure 35 in annex). Here, the LSTM algorithm is not used in a usual deep learning context, where no feature selection is needed, but rather as a regular classifier similar to Random Forest or SVM.

I obtained a training an accuracy of 0.9006 – and a validation accuracy of 0.8913, with ReLu activation function.

However, classifier was not able to diverge or even enhance performance with Sigmoid as activation function, with accuracy hovering over 0.5 and loss hovering over 0.6932.

Results show that the performance of LSTM on featured data is very close to that of standard ML classification algorithms, however at a cost of much higher number parameters.

LSTM is a deep learning model that extracts features automatically, and thus no prior feature selection is needed. In the next chapter of the implementation, I apply LSTM on raw data in a deep learning context (without feature selection) and then I evaluate the results. I then apply it to kmer encoded sequences. Results are presented in the same chapter, in a section titled "LSTM on kmer encoded sequences".

MSE MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

Deep Learning based Classification using

LSTM

iii. LSTM

Overview:

In the last chapter, I explained the implementation of DNA sequence classification using classical ML algorithms on a Bag of Words model (used in NLP), where the features are kmers and the data is the frequency of each kmer within each sequence. The approach has demonstrated a relatively good classification capability.

In this chapter, I experiment and present the implementation of a classification with deep learning, using LSTM.

Recall - RNNs:

For humans, thoughts have persistence, where they use reasoning about previous events to take an action at the certain moment. Similarly, and unlike traditional neural networks, Recurrent Neural Networks (RNNs) can - in theory - take a context into consideration to perform a task (in our case classification). However, in practice, RNNs are only good in short term context, where only recent information is retained.

LSTM, short for Long-Short-Term-Memory, addresses this problem. LSTMs are a type of RNNs capable of learning long-term dependencies. Remembering information for long periods of time is practically their default behavior. (Christopher Olah 2015). LSTM takes sequential data as input and gives sequential, categorical, or binary output.

VIII.iii.1 LSTM Pipeline

Figure 32 below shows the general LSTM pipeline.

Tesorflow's implementation of LSTM requires that sequences have the same length, so I used padding. (PyTorch allows for variable length with RNNs, however this was discovered later and is out of scope of this study). The padding used is zero padding, post-sequence (to maintain the requirements of Cuda library).

The sequences are then encoded into numerical values using sequential encoding technique (click to see Table 2).

Due to memory limitations for model training, (on both the local machine and online resources (google colabs)), I implemented two experiments. In the first, I truncated the sequences to a shorter length (7500), and for the second, I kept the full sequences length, but needed to split the dataset into chunks of samples (5 chunks) then I updated the model by training it on each chunk. To simulate the epochs on the split dataset, the whole training pass is repeated several times as an alternative for epochs, or otherwise repeated on each chunk with only one pass on the whole dataset. Model needs to be trained using both methods, to measure the difference.

Keras Embedding layer gives the possibility to ignore padding from learning, by using the mask_zeros option. Model needs to be trained with and without masking, to measure the difference.

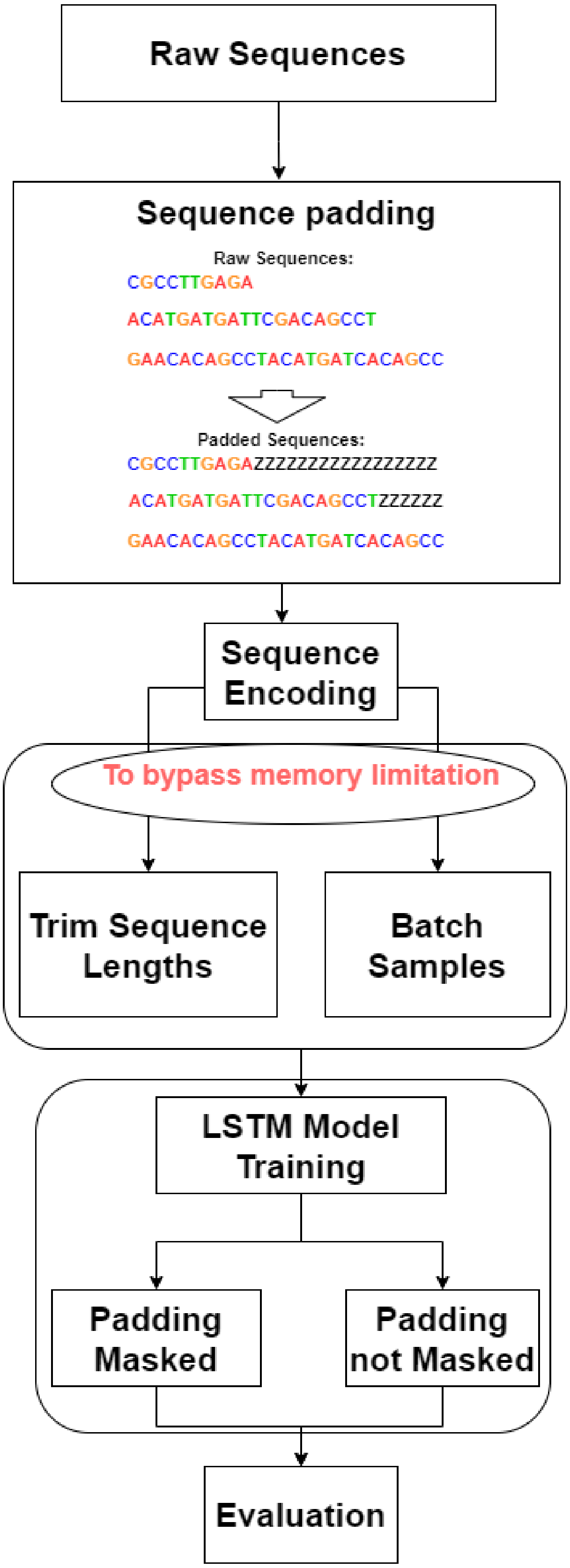


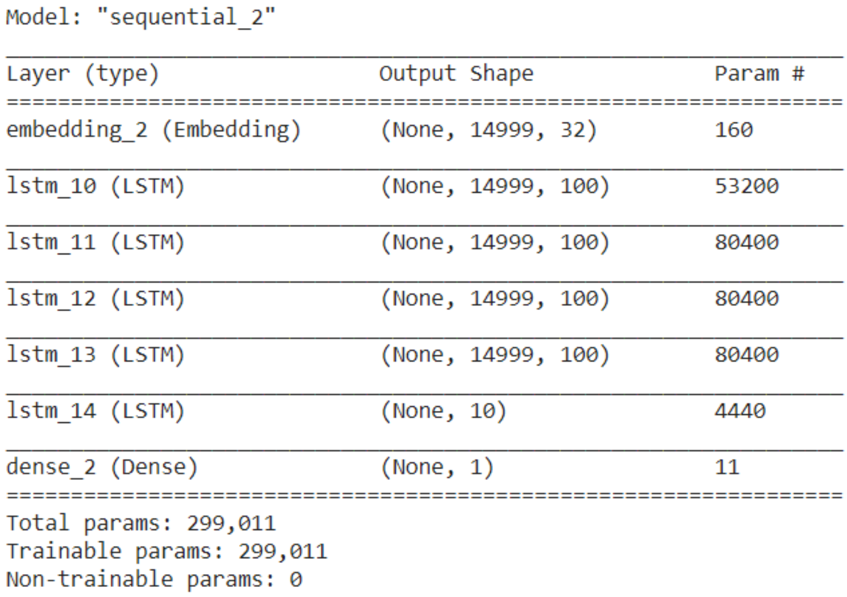
Figure 32 General LSTM Pipeline

VIII.iii.2 Training and evaluation

There is no definite rule on how many nodes or how many layers one should choose for LSTM, instead, a trial-and-error approach is used. So first I trained a simple one-LSTM-layer model and obtained very bad performance (50% accuracy).

I then increased the number of layers for the network to be deep. I thus created a five-layers LSTM model (Figure 33), with:

- epochs=3, batch_size=64
- embedding vector length=32
- activation='sigmoid'
- optimizer='adam'
- metrics='accuracy'
- loss='binary_crossentropy' :



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 14999, 32)         160
_________________________________________________________________
lstm_10 (LSTM)               (None, 14999, 100)        53200
_________________________________________________________________
lstm_11 (LSTM)               (None, 14999, 100)        80400
_________________________________________________________________
lstm_12 (LSTM)               (None, 14999, 100)        80400
_________________________________________________________________
lstm_13 (LSTM)               (None, 14999, 100)        80400
_________________________________________________________________
lstm_14 (LSTM)               (None, 10)                4440
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 11
=================================================================
Total params: 299,011
Trainable params: 299,011
Non-trainable params: 0
```

Figure 33 LSTM deep model

Then I trained the model on four different combinations, with full and truncated sequences, with and without padding. Results obtained are shown in Table 14 as follows:

Sequences	Padding Mask	Loss	Accuracy	Validation accuracy
Truncated Sequences	Yes	**0.6932**	**0.5050**	**0.5044**
	No	**0.6932**	**0.4982**	**0.4956**
Full Sequences	Yes	**0.6932**	**0.4988**	**0.5067**
	No	**0.6934**	**0.5123**	**0.5042**

Table 14 Results of LSTM training on both truncated and full sequences, with and without padding mask.

Analysis of results:

Results show that the highest achieved train and validation accuracy is about 50%, which means that training accuracy is not better than the validation accuracy. Also, the loss is not decreasing. Thus, padding, and truncating sequences had no effect, whether positive or negative on the outcome.

To mitigate the bad performance, the following actions have further been taken, as shown in Table 15 below:

Action taken	Details	Result

Tune learning rate hyper-parameter	Tried different values for the
	`learning_rate :[0.1,0.01,0.001,0.0001]`
	`opt = keras.optimizers.Adam(learning_rate)`
		No improevment

Increase the number of LSTM units and layers	LSTM units:100 LSTM layers: 7	Session crashes
Check target variable distribution	Made sure that the distribution of the target variable is uniform among train/validation/test sets	No improvement
Use repeat-padding instead of zero padding	It means in short sequences, that instead of filling the rest of the sequence with 0s, repeat the same sequence to fill all those empty places.	No improvement
Truncated sequences to shortest sequence length	To completely remove the effect of padding	No improvement
Use Numpy arrays instead of python lists	It has been shown that they are 4 times faster.(Babenhauserheide 2014)	No improvement
Used one hot encoding instead of sequential encoding.	It means that ['A', 'C', 'G', 'T'] becomes [[1,0,0,0],[0,1,0,0,],[0,0,1,0],[0,0,0,1]]	No improvement
Used ReLu activation function instead of Sigmoid in the dense layer		No improvement

Table 15 Actions taken to improve simple LSTM model bad performance.

iv. LSTM on kmer encoded sequences.

Overview:

In the last section, I introduced the procedure and steps taken to implement classification of raw sequences with LSTM network. Accuracy was very bad (0.5). To take the research further, I wanted to test LSTM on kmer-encoded sequences, based on assumption that a longer word would change the results. The kmer encoding is the same as the one used in the NLP/BoW technique, i.e. overlapping kmers.

VIII.iv.1 LSTM on kmers Pipeline

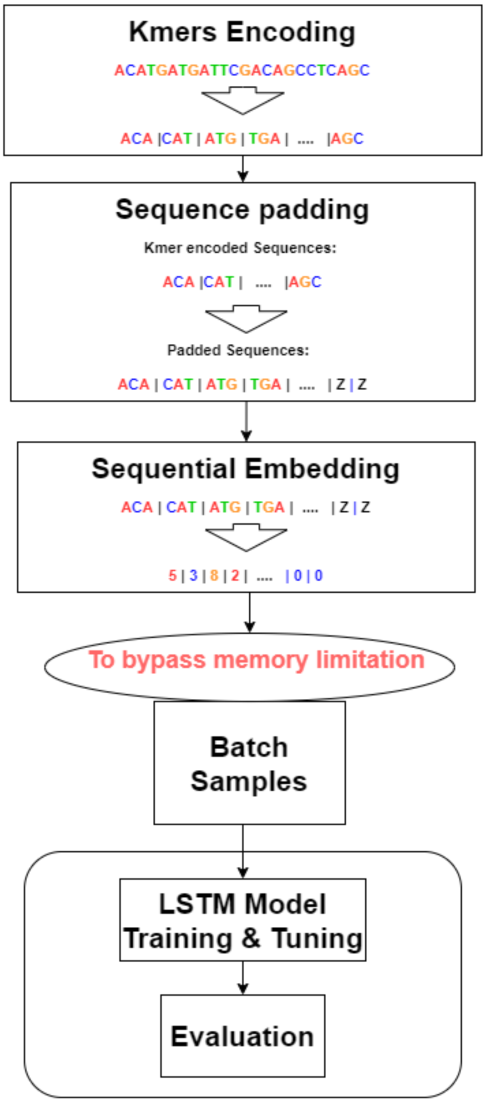


Figure 34 LSTM on Kmer-encoded sequences pipeline. Here sequences are first padded, then kmer-encoded, before being fed into a deep LSTM model.

VIII.iv.2 Training and evaluation

I encoded the raw sequences as kmers of size 3 (not to confuse with the BoW model, where the counts of kmers where taken). I then zero-padded the sequences to maintain a uniform sequence length, as required by LSTM. Then, I embedded the kmer encoded (now padded) sequences into integer representation, using *keras one_hot method: (*`from keras.preprocessing.text import one_hot`*).*

With sequences ready to be fed to training, I used the same simple LSTM model as before. I modified the model progressively depending on the results, by increasing the number of LSTM units and the depth of the network. Data is split in a stratified fashion, into training and test sets with10% test ratio.

Analysis of results

Three epochs of training were run over each chunk of the dataset. The training and validation accuracies were both near 0.5, with a steady loss of 0.693, for both training and validation. Figures Figure 36, Figure 37, Figure 38 and Figure 39 in the annex, show how steady the variation of training & validation, and evaluation (test) accuracies and loss.

Another run took place, this time with three epochs over the whole dataset (repeated training over the chunks), and no difference in result was observed.

To maintain comparability, I have taken similar actions to those mentioned in Table 15 above, none of which have demonstrated contribution to the improvement of the performance.

It is reasonable to conclude that the reason for the bad performance is not the model itself. The research must proceed upstream, inspecting the data itself, or use another library instead of TensorFlow, where variable length sequences are allowed. At this point, the assumptions are tested, and the research algorithm has halted.

Conclusions & Recommendations

IX. Results, Conclusions and Discussions

This research had the objectives of establishing an inventory of machine learning techniques suitable for processing genomic sequences, then selecting implementing and documenting these methods, then testing these methods on data sets of increasing complexity and difficulty, and finally evaluating them in terms of the predictive power and computation cost and time.

.

Along with the general objectives, the research has three specific objectives, corresponding to three levels of complexity of the data on which classification models needed to be tested. First, classification needed to be validated on relatively short artificial DNA sequences (between 5000 and 15,000 character long). The second objective is to validate the scalability of the models on longer real DNA sequences of both viruses and bacteria apart. The third objective was to develop a model capable of classifying real sequences of heterogenous types (virus and bacteria).

The general objectives have all been met, where an inventory of techniques for every step in the pipeline has been established and the methods for data preprocessing and classification have been created, trained, tested, and documented.

Two learning approaches were experimented on, which are: Classical machine learning and Deep Learning.
Classical machine learning approach was the Natural Language Processing approach, specifically the Bag of Words model (kmers and their counts for each sequence) and Deep Learning is based on LSTM; a Recurrent Neural Network model capable of dealing with long sequences due to its long-term memory feature.

Among the presented classification approaches, the classical ML/NLP based method demonstrated a relative success, with AUC score of ~93%. The methods of data preprocessing for the classical machine learning approach were very time consuming, however necessary to extract features needed for the model to learn.

Feature selection has demonstrated an important role in improving the performance of classification with classical ML/NLP model. Feature selection equally allowed to discover which are the sub-sequences that allowed the classification.

Deep learning models however does not necessarily need the feature extraction step. Given that the raw sequence is long, we were interested in LSTM/RNN deep learning model, known for long-term memory. LSTM experimentation was however unsuccessful, with no learning taking place due to stable loss over training epochs (accuracy 50%). Many actions have been taken to mitigate this, such as adjusting the depth of the network, the size of each layer, learning steps, activation functions and even an extra-feature extraction has been introduced by feeding in a sequence of kmers instead of a raw sequence, none of which has been able to change the outcome.

It is still early to draw conclusions on the potential of LSTM in sequence classification (5000->15000 character long), where an upstream research is required on LSTM's capacity before drawing a solid conclusion. Therefore, before a deeper analysis of the reasons of failure takes place, one cannot decide whether LSTM is promising and which of these approaches seem to be most promising for sequence classification.

Both approaches were able to function (on the dataset of 60k of artificial sequences), however with LSTM not yet successful, they were not tested on bigger datasets, due to hardware and time limitations.
Poor performance demonstrated with LSTM may be due to the nature of the data itself, which is artificially created.

Some online examples (Brownlee 2016) showed that LSTM demonstrated good classification results, however with short sequence of words (around 500 words), whereas in this experiment, sequences are of order of thousands of words (word = single character for LSTM model or word = kmer, for the third LSTM sub-model: LSTM on Kmers model).

It was not possible to conclude how do these methods behave with mixed sequence types (a mix of viruses plus bacteria DNA sequences as input), as this step was not reached.

Limitations and difficulties:

Resources in hand for this research were a personal computer with limited RAM (8GB) and a ML incompatible GPU. Certain tasks took a much longer time than expected (Kmer extraction, kmer counting). Other tasks were not feasible at all, such as running the DNA2VEC embedding method, which failed due to dependency on libraries that have issues with the hardware.

A GPU would have accelerated the training and testing of LSTM based models. A bigger RAM and more CPU cores would have accelerated the data preprocessing and training.

The lack of prior experience with RNNs/LSTM and the lack of time required to acquire deep knowledge required to analyze the reasons for the failure of LSTM, and the lack of online examples of LSTM with long sequences were all difficulties.

Using TensorFlows's LSTM had its drawbacks, and that is because sequences need to be of the same length, requiring either padding sequences, thus introducing noise, or truncating sequences, thus eliminating potential information. PyTorch is an alternative that allows variable length sequences; however, time did not allow to experiment on it.

Opportunities

This project allowed me to execute an end-to-end applied research. It helped me build practical competencies and develop a taste for bio-informatics. The project management aspect was very enriching, where I needed to overcome my perfectionism and deal with objectives in terms of deadlines and priorities.

Recommendations

- Test on protein sequences (20 letter alphabet).
- Inspect in depth, the embedding layer for LSTM, to decide on the best embedding for the problem in hand.
- Try PyTorch instead of TensorFlow for LSTM on long sequnces, as it is known to be faster with RNN and allows for variable length sequences.
- Use python directly with an IDE due to the memory requirements of ipython notebooks.
- Explore other programming language allowing faster manipulation of data structures than python.
- Try DNA2Vec as an embedding method allowing to measure dissimilarity.

Further research

- Explore information theory-based Alignment-Free (AF) methods of DNA comparative analysis
- Explore information-theoretical-based Feature Selection methods. These methods usually require a lot of computing power and can be slow on ordinary machines, like the one used in this project.

X. References

Aman1608. 2020. “Feature Selection Techniques in Machine Learning.” https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/.

Babenhauserheide, Arne. 2014. “Memory Requirement of Python Datastructures.” https://www.draketo.de/english/python-memory-numpy-list-array.

Brownlee, Jason. 2016. “Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras.” https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/.

Charfaoui, Younes. 2020. “Hands-on with Feature Selection Techniques: Embedded Methods.” https://heartbeat.fritz.ai/hands-on-with-feature-selection-techniques-embedded-methods-84747e814dab.

Choong, A C H, and N K Lee. 2017. “Evaluation of Convolutionary Neural Networks Modeling of DNA Sequences Using Ordinal versus One-Hot Encoding Method.” In *2017 International Conference on Computer and Drone Applications (IConDA)*, , 60–65.

Christopher Olah. 2015. “Understanding LSTM Networks.” https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Du, Jingcheng et al. 2019. “Gene2vec: Distributed Representation of Genes Based on Co-Expression.” *BMC Genomics* 20(1): 82. https://doi.org/10.1186/s12864-018-5370-x.

Google. “Machine Learning Crash Course.” https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

Jason Brownlee. 2018. “A Gentle Introduction to Normality Tests in Python.” https://machinelearningmastery.com/a-gentle-introduction-to-normality-tests-in-python/.

———. 2020. “How to Choose a Feature Selection Method For Machine Learning.” https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/.

Julia Kho. “Why Random Forest Is My Favorite Machine Learning Model.” https://towardsdatascience.com/why-random-forest-is-my-favorite-machine-learning-model-b97651fa3706.

Kimothi, Dhananjay, Akshay Soni, Pravesh Biyani, and James M Hogan. 2016. “Distributed Representations for Biological Sequence Analysis.”

Ng, Patrick. 2017. “Dna2vec: Consistent Vector Representations of Variable-Length k-Mers.”

Scikit-learn. “Cross-Validation: Evaluating Estimator Performance.” https://scikit-learn.org/stable/modules/cross_validation.html#.

———. “Stochastic Gradient Descent.” https://scikit-learn.org/stable/modules/sgd.html#:~:text=SGDClassifier supports multi-class classification,that and all other classes.

Sivasai Yadav Mudugandla. 2020. “10 Normality Tests in Python.” https://towardsdatascience.com/normality-tests-in-python-31e04aa4f411.

Wikipedia contributors. 2021a. “Accuracy Paradox --- {Wikipedia}{,} The Free Encyclopedia.” https://en.wikipedia.org/w/index.php?title=Accuracy_paradox&oldid=1022507114.

———. 2021b. “Deep Learning --- Wikipedia The Free Encyclopedia.” https://en.wikipedia.org/w/index.php?title=Deep_learning&oldid=1022252019.

———. 2021c. “FASTA Format --- Wikipedia The Free Encyclopedia.” https://en.wikipedia.org/w/index.php?title=FASTA_format&oldid=1021800055.

Yang, Aimin et al. 2020. “Review on the Application of Machine Learning Algorithms in the Sequence Data Mining of DNA .” *Frontiers in Bioengineering and Biotechnology* 8: 1032.

https://www.frontiersin.org/article/10.3389/fbioe.2020.01032.

Zielezinski, Andrzej et al. 2019. “Benchmarking of Alignment-Free Sequence Comparison Methods.” *Genome Biology* 20(1): 144. https://doi.org/10.1186/s13059-019-1755-7.

Zielezinski, Andrzej, Susana Vinga, Jonas Almeida, and Wojciech M Karlowski. 2017. “Alignment-Free Sequence Comparison: Benefits, Applications, and Tools.” *Genome Biology* 18(1): 186. https://doi.org/10.1186/s13059-017-1319-7.

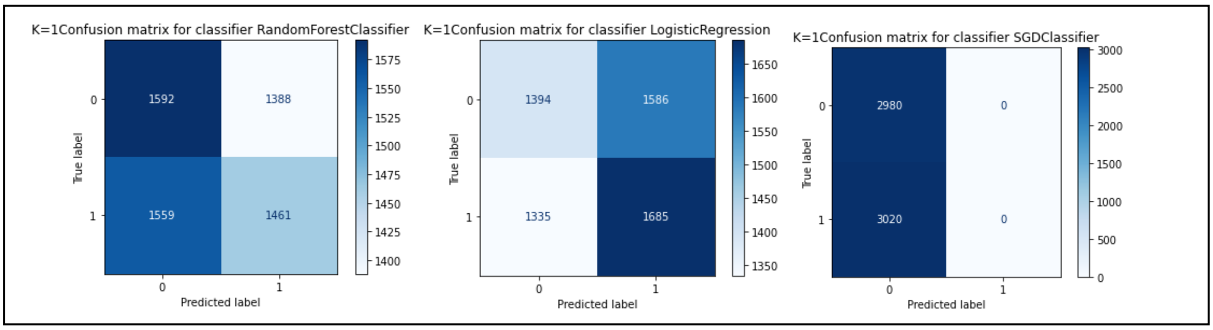
XI. Annexes

Reference with k=1

The results with **k=1:**

Table 16 Reference classifiers training results, K=1

Classifier	accuracy	precision	recall	f1-score	Training time
RandomForestClassifier	0.509	0.509	0.509	0.509	15 secs
LogisticRegression	0.513	0.513	0.513	0.512	<1 secs
SGDClassifier	0.548	0.581	0.548	0.501	1 secs

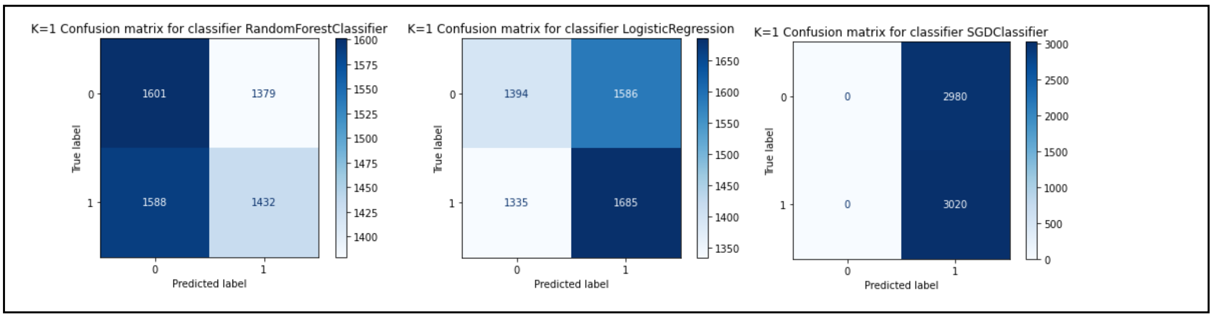


Reference with k=2

The results with **k=2:**

Table 17 Reference classifiers training results, K=2

Classifier	accuracy	precision	recall	f1-score	Training time
RandomForestClassifier	0.507	0.507	0.507	0.507	21 secs
LogisticRegression	0.509	0.509	0.509	0.509	< 1 secs
SGDClassifier	0.514	0.527	0.514	0.436	3 secs



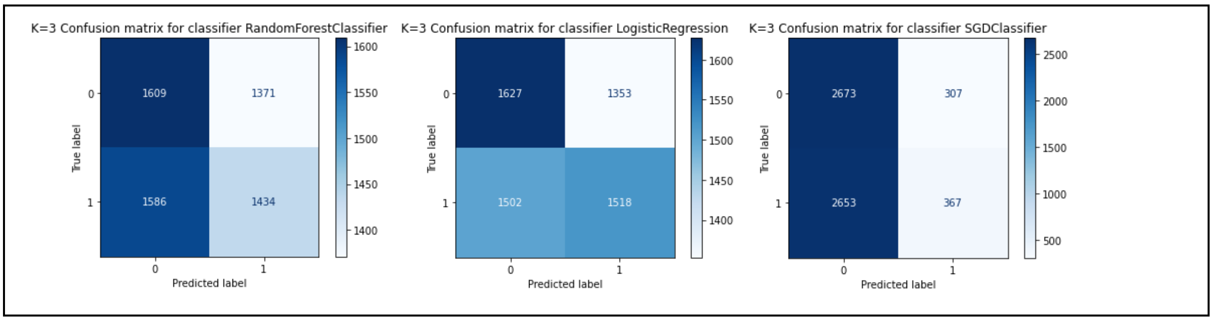
.

Reference with k=3

The results with **k=3:**

Table 18 Reference classifiers training results, K=3

Classifier	accuracy	precision	recall	f1-score	Training time
RandomForestClassifier	0.507	0.507	0.507	0.507	30 secs
LogisticRegression	0.524	0.524	0.524	0.524	1 secs
SGDClassifier	0.507	0.523	0.507	0.42	5 secs

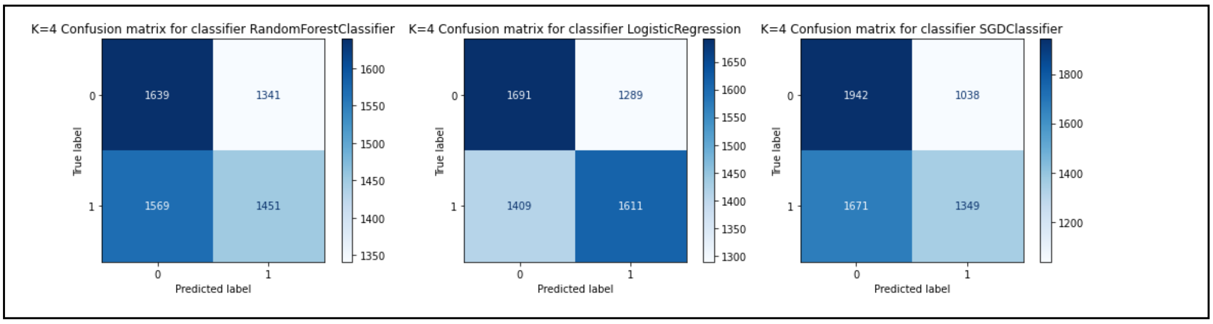


Reference with k=4

The results with **k=4:**

Table 19 Reference classifiers training results, K=4

Classifier	accuracy	precision	recall	f1-score	Training time
RandomForestClassifier	0.515	0.515	0.515	0.514	52 secs
LogisticRegression	0.550	0.551	0.550	0.550	2 secs
SGDClassifier	0.548	0.551	0.548	0.544	51 secs

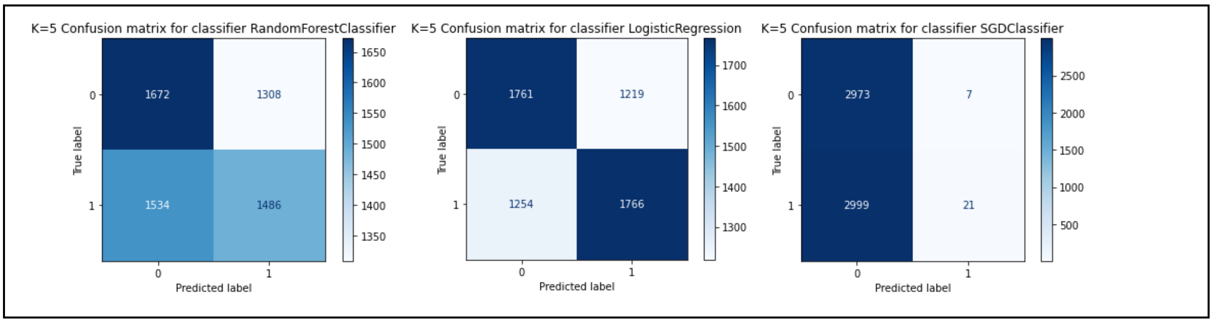


Reference with k=5

The results with **k=5:**

Table 20 Reference classifiers training results, K=5

Classifier	accuracy	precision	recall	f1-score	Training time
RandomForestClassifier	0.526	0.526	0.527	0.526	78 secs
LogisticRegression	0.588	0.588	0.588	0.588	6 secs
SGDClassifier	0.499	0.625	0.499	0.337	51 secs

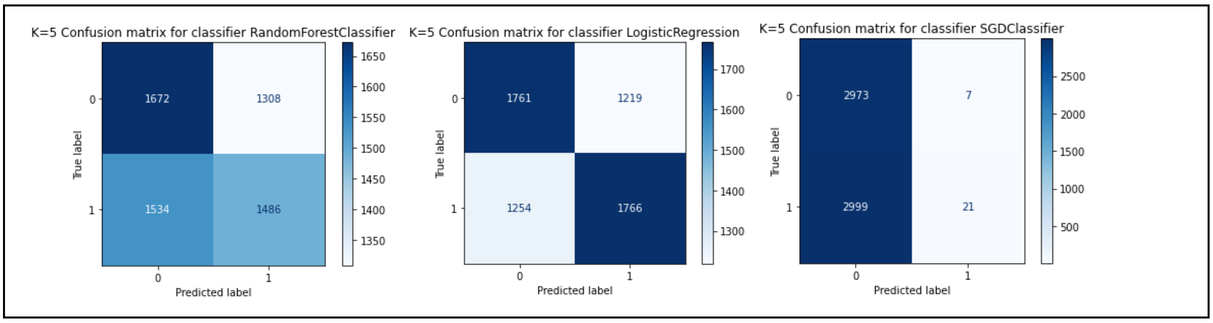


Reference with k=6

The results with **k=6:**

Table 21 Reference classifiers training results, K=6

Classifier	accuracy	precision	recall	f1-score	Training time
RandomForestClassifier	0.582	0.582	0.582	0.582	121 secs
LogisticRegression	0.655	0.655	0.655	0.655	23.7 secs
SGDClassifier	0.535	0.681	0.535	0.415	120 secs
Discarded: ~~LinearSVC~~	~~0.644~~	~~0.654~~	~~0.644~~	~~0.6~~	~~4057 secs~~

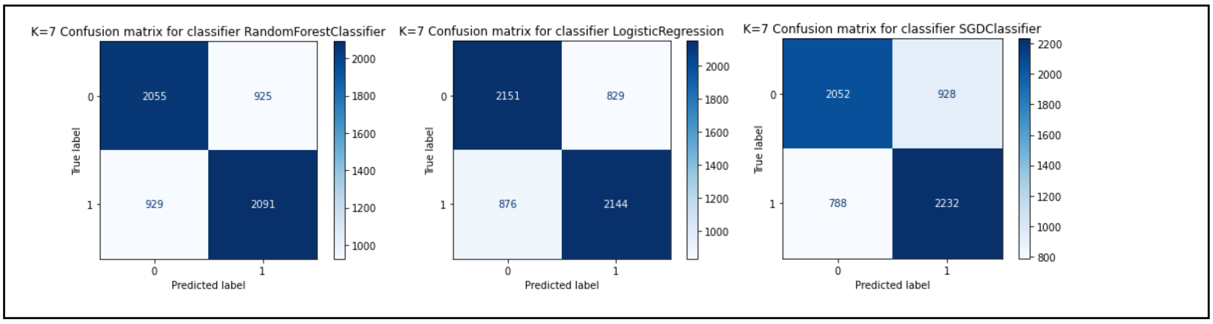


Reference with k=7

The results with **k=7:**

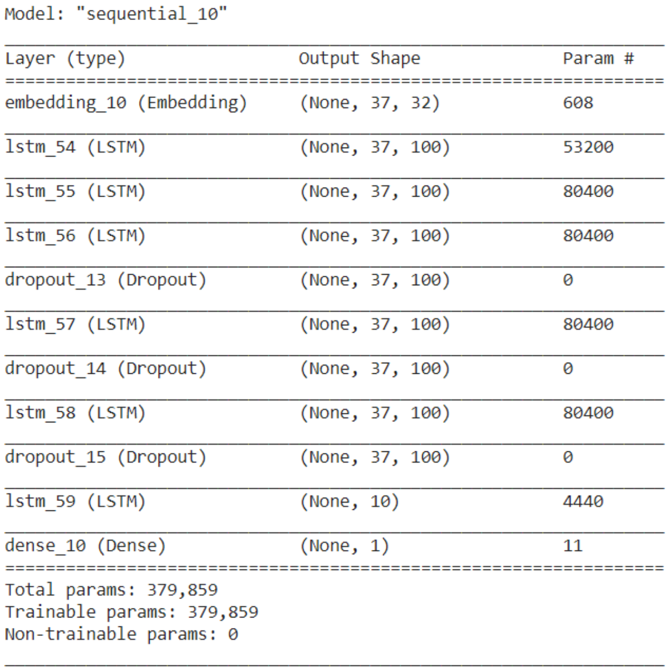
Table 22 Reference classifiers training results, K=7

Classifier	accuracy	precision	recall	f1-score	Training time
RandomForestClassifier	0.691	0.691	0.691	0.691	211 secs
LogisticRegression	0.716	0.716	0.716	0.716	157 secs
SGDClassifier	0.714	0.714	0.714	0.714	544 secs



LSTM model

Deep LSTM network with 6 layers, five with100 units each, and 10 units for the last layer.



```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_10 (Embedding)     (None, 37, 32)            608       
_________________________________________________________________
lstm_54 (LSTM)               (None, 37, 100)           53200     
_________________________________________________________________
lstm_55 (LSTM)               (None, 37, 100)           80400     
_________________________________________________________________
lstm_56 (LSTM)               (None, 37, 100)           80400     
_________________________________________________________________
dropout_13 (Dropout)         (None, 37, 100)           0         
_________________________________________________________________
lstm_57 (LSTM)               (None, 37, 100)           80400     
_________________________________________________________________
dropout_14 (Dropout)         (None, 37, 100)           0         
_________________________________________________________________
lstm_58 (LSTM)               (None, 37, 100)           80400     
_________________________________________________________________
dropout_15 (Dropout)         (None, 37, 100)           0         
_________________________________________________________________
lstm_59 (LSTM)               (None, 10)                4440      
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 11        
=================================================================
Total params: 379,859
Trainable params: 379,859
Non-trainable params: 0
_________________________________________________________________
```

Figure 35 LSTM modes, used as a classifier after feature-selection, to establish a benchmark with Classical ML algorithms.

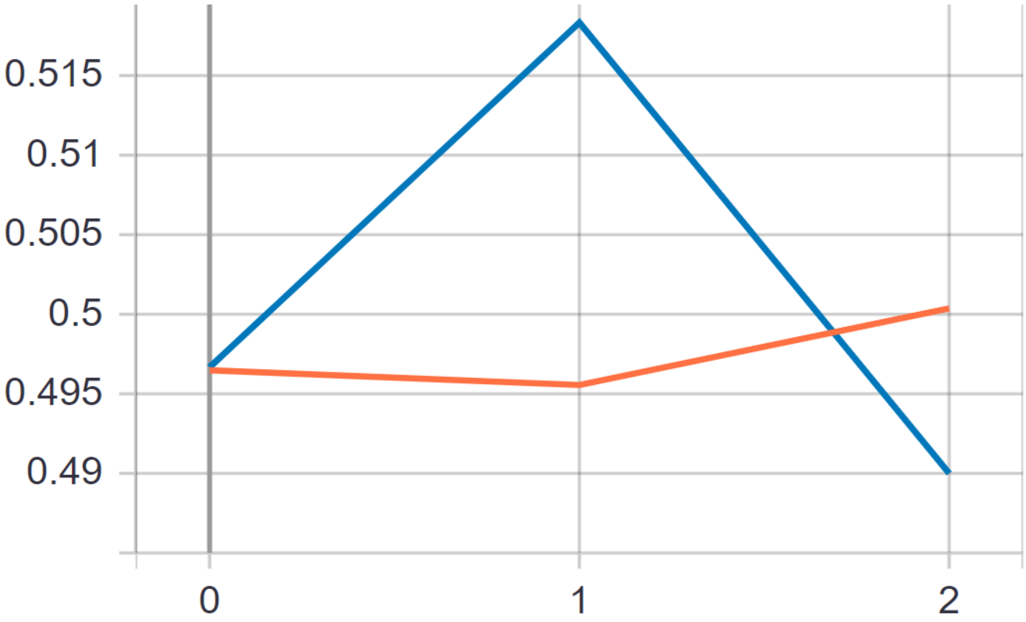


Figure 36 ***Training*** *and* ***validation*** *accuracy of LSTM on Kmers with sigmoid activaiton, over three epochs. Both training and validation accuracies swing around 0.5.*

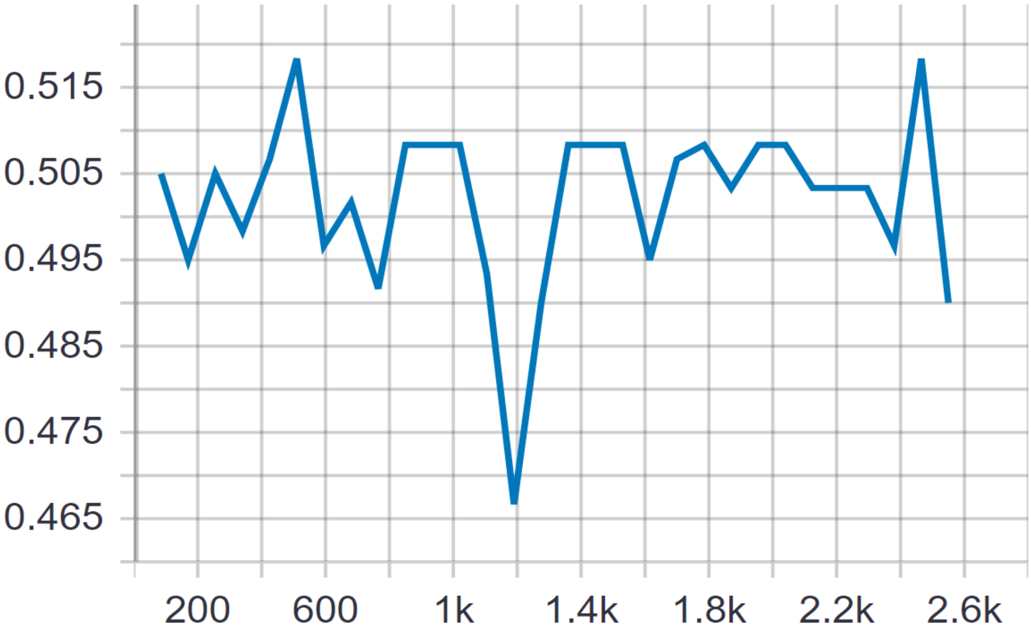


Figure 37 Evaluation (Testing) ***accuracy*** *of LSTM on Kmers with sigmoid activaiton versus number of iterations, swinging around 0.5.*

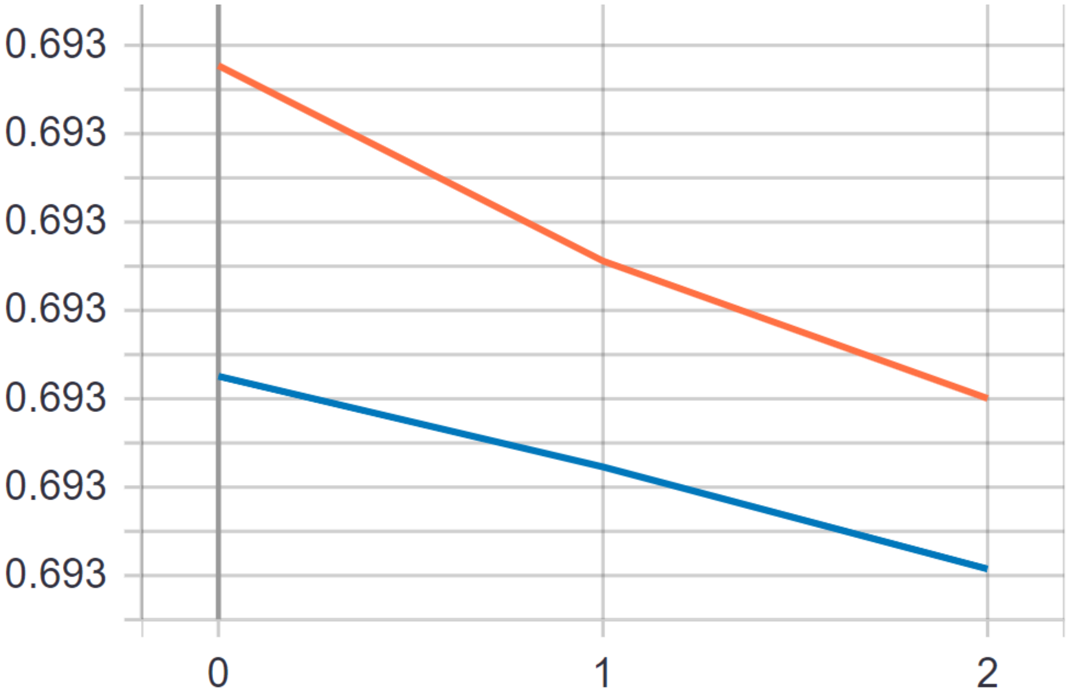


Figure 38 ***Training*** *and* ***validation*** *loss, exhibiting slight decrease.*

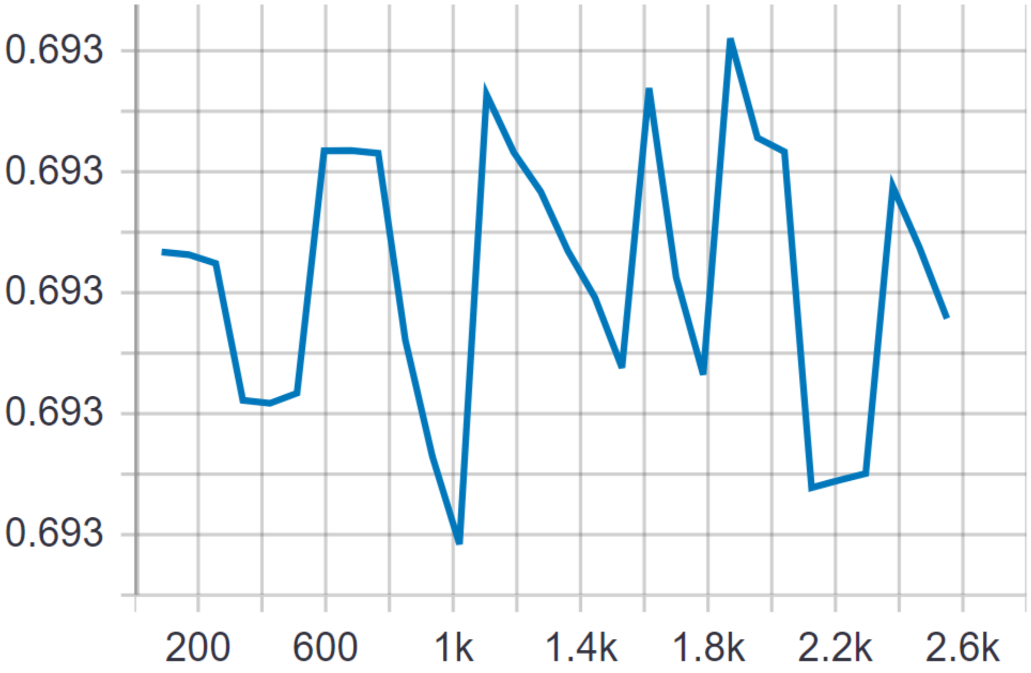


Figure 39 Evaluation (Testing) ***loss*** *versus number of iterations, swinging around 0.693*

XII. Source Code

The source code of the developed scripts / notebooks can be found on GitHub at the following link.

Link to Git-Hub repository

https://github.com/khansabassem/MLonGenomics

There are three types of scripts:

1. Jupiter notebooks ipython format.
2. Python scripts (for early, parallelized kmer counting methods developed using libraries that are incompatible with Jupiter notebooks).
3. Bash scripts used with old/early methods to:
    a. Run KMC kmer counter.
    b. Optionally delete intermediary files resulting from the three preprocessing phases.

Directories:

There are three main directories:

1. BoW: notebooks & scripts of the first ML approach
2. LSTM: notebooks of the LSTM approach.
3. LSTM on KMERS: notebooks of the LSTM on KMERS approach
4. Data: contains the raw sequences data.