MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Analysis and classification of long terminal repeat (LTR) sequences using machine learning approaches

Master's Thesis

JAKUB HORVÁTH

Brno, Spring 2023

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Analysis and classification of long terminal repeat (LTR) sequences using machine learning approaches

Master's Thesis

JAKUB HORVÁTH

Advisor: Ing. Matej Lexa, Ph.D.

Department of Machine Learning and Data Processing

Brno, Spring 2023


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Jakub Horváth

Advisor: Ing. Matej Lexa, Ph.D.

Acknowledgements

I would like to thank my advisor Ing. Matej Lexa, Ph.D. for his continued support and guidance throughout this long process, as well as the team from the Institute of Biophysics in Brno, who have provided me with materials and information whenever necessary.

Abstract

This thesis focuses on the analysis and classification of long terminal repeat (LTR) sequences, which are critical components of retrotransposons that play a significant role in genome structure and evolution. The work employs frequent pattern-mining techniques to identify significantly co-occurring motifs in LTR sequences, with the goal of characterizing their diversity and distribution.

For the classification task, three machine-learning models have been trained with the goal of recognizing LTR sequences based on extracted features and raw sequential properties. These models include the Gradient Boosting classifier, neural networks, and the BERT transformer, which are trained and tested on a dataset of annotated LTR sequences. The analysis and classification results demonstrate the utility of frequent pattern mining for identifying important motifs in LTR sequences, as well as the effectiveness of different machine-learning models in the task of DNA sequence classification. Further analysis of the trained models provides insight into LTR sequences' structure and potential function.

Keywords

bioinformatics, large-scale data processing, machine learning, deep learning, feature analysis, DNA sequence analysis, neural networks, large language models

Contents

List of Tables

List of Figures

1 Introduction

Long Terminal Repeats (LTRs) are identical or nearly identical sequences located at the ends of retrotransposons and retroviruses that play essential roles in retrotransposon mobility and gene regulation. Initially thought to be parasitic elements, LTR sequences are now known to be important building blocks of the transcriptional regulatory collection, as the host genome has co-opted them to regulate its gene expression. [1]

LTRs are created during reverse transcription of the original LTR-retrotransposon when the reverse transcriptase enzyme responsible for transposon replication reaches the end of the RNA template and transfers to the other end of the RNA molecule, copying the LTR sequence from the opposite end of the RNA[2]. This results in the duplication of the LTR sequence at both ends of the element.

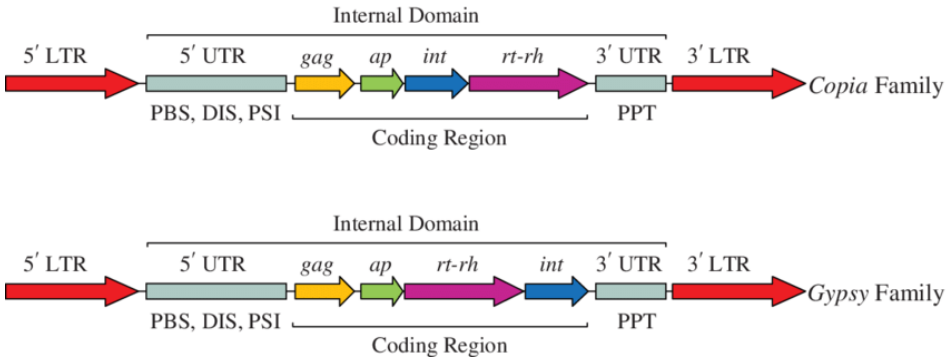

Figure 1.1: LTR position with repect to the element [3]

LTRs contain sequences that can regulate the expression of the retrotransposon, such as promoters, enhancers, and repressors.

Their sequences have been found to be present in various organisms, including plants, animals, and fungi, and have been shown to play essential roles in regulating stress-responsive genes in plants. In animals, LTRs have been implicated in the regulation of immune response genes.

Analysis and classification of LTR sequences are crucial for understanding their biological functions. However, identifying and charac-

terizing LTRs can be challenging due to their repetitive nature and high sequence similarity with other genomic sequences. In this thesis, I aim to analyze and classify LTR sequences in multiple plant species, as LTR-retrotransposons and their accompanying LTRs are highly dominant in plants due to evolutionary processes and genetic pathways linked to cellular stress[4]. Various computational tools and techniques have been developed to address these challenges and to enable the analysis of LTR sequences. Specifically, the work focuses on building a reliable database, classifying these sequences, and analyzing their internal structure using various model interpretation techniques.

Model interpretation is currently a large area of interest, as with the growing complexity of large machine-learning models grows the need for their understanding. By understanding these models, we may be able to comprehend the decisions they make, as well as the data itself.[5]

The findings should provide insight into the diversity, distribution, and evolution of LTR sequences in the analyzed species and their further connection to plant transcriptional regulatory processes.

2 Database Preparation

2.1 Positive Training Database

The basis of any data analysis and machine learning task is the data itself, as low-quality training data is bound to produce inaccurate models.

The training database preparation was conducted in 3 steps:

1. Obtaining LTR-TEs from publicly available databases
2. Extracting LTR sequences from the TEs
3. Removing redundant elements using a clustering technique

In the first step, sequences are obtained from the following databases:

DB name	N. elements	Element type
GyDB[6]	253	LTRs
Soybase[7]	31183	LTR-retrotransposons
TREP[8]	1297	LTR-retrotransposons
InpactorDB[9]	52616	LTR-retrotransposons
GrTEdb[10]	12231	LTR-retrotransposons

Table 2.1: TE Databases used

Next, the LTR sequence is extracted. This can be done in two ways: either by aligning the opposite ends of the transposable element and extracting the matching sequence region (as these are identical or highly similar) or by parsing the sequences using state-of-the-art tools for transposable element recognition and annotation. The latter option is used in my final work as it is faster and provides a way to re-verify the downloaded sequences and remove any elements that the database's creators could have misclassified. The tools used for this were LTRFinder[11] and LTRHarvest[12]. These two often complement each other in TE sequences detected, where some elements

recognized by the first may not be recognized by the second, and vice-versa. Therefore, a union of the detected sequences was taken, where the longer version of a sequence was accepted into the final dataset if both tools had recognized it.

This method was chosen because LTRFinder had been found to cut the ends of LTRs, which could potentially result in a loss of information in the training set.

Sequence clustering was run on the dataset in the last step to detect highly similar sequences. For this, the program CDHIT[13] was used.The tool first removes low-quality sequences, then compares each sequence to all others by breaking them down into smaller chunks called words and comparing these. Sequences are clustered based on this similarity if surpassing a given threshold.

In my case, I used the threshold of **89%** similarity, which proved efficient at eliminating nearly identical sequences.

The output of CDHIT was parsed using a script by J. Healey[14] for parsing outputs of the program, which groups together sequence ids, making their extraction from sequence files easier.

2.2 Negative Training Database

The negative training database was slightly more tricky to obtain, as no clear DNA sequence candidate can be used as a counterexample to the LTR. LTRs, essentially regulatory regions, share many similarities with commonplace enhancers and promoters, which could prove problematic when training the models.

Three approaches were implemented for generating negative sequences:

1. Random sequence generation
2. Genomic sequence extraction
3. Markov chain generation

2.2.1 Random sequence generation

This process includes randomly creating sequences in the form $(b_1, ...b_n)$ where b_i is one of the four bases (A, C, T, G), sampled from a uniform

probability distribution with $p = 0.25$ and n is a sequence length sampled from the distribution of LTR sequence lengths in the positive training set 2.2. Although this process is the simplest algorithmically and computationally, it tends to result in falsely high accuracy measures on the training and validation sets. When tested on another set, where non-LTR sequences were genomic sequence extracts and false positive sequences misclassified by TE recognition tools, the model's precision measure ($\frac{TP}{TP+FP}$) drastically decreased. This was because the randomly generated sequences were far too easy to distinguish from actual DNA sequences. The model overfitted on motifs that tend to be fairly common in actual DNA sequences but uncommon in the randomly generated ones, where each base had the same probability of being chosen. This is rarely the case in genomic sequences, as even random genomic regions tend to contain certain patterns.

Despite this fact, however, the random sequence set was initially used for prototyping trained models, and it was later found that models trained using negative training sets consisting of roughly **25%** random sequences achieved the best results.

2.2.2 Genomic sequence extracts

The genomic sequences were introduced after random sequences proved inadequate to be used by themselves in training. These sequences come from the genomes of 5 different plant species: Casuarina equisetifolia[15], Citrullus lanatus[16], Hordeum vulgare[17], Juglans regia[18] and Zea mays[19].

Once again, the length of the sequences was chosen to mimic the length distribution of the actual LTRs. It was necessary to avoid LTR regions in the selected genomes and prevent the inclusion of an LTR sequence labeled as a non-LTR, as this could negatively impact the training. For this purpose, I used an annotation[20] of these five genomes, avoiding the extraction of regions marked as LTRs and low-quality sequencing regions (large segments containing "N"), which, as I later found, contributed to a large part of the downloaded data, spanning entire chromosomes. These sequences had to be resampled with a constraint on the "N" content to avoid assembly/sequencing errors affecting the results. Genomic sequence extracts represent around **60%** of the final negative training set.

2.2.3 Markov chain generated sequences

In order to try to create a sequence that mimics LTRs, but is not an actual LTR, a more complex algorithmic approach was further tested. This was attempted in order to learn the deeper relations of sequential motifs contained in LTRs and prevent overfitting on 2-mers and 3-mers common in DNA. As seen before, randomly generated sequences do not force the algorithm to learn the more intrinsic properties of LTRs. To avoid this issue, Markov chains were used to generate artificial sequences.

Markov chains are a mathematical tool used to model a sequence of events, where the probability of each event depends only on the outcome of k previous events.

These models are useful for generating artificial DNA sequences because they can be trained to model the probability of nucleotide transitions in a given sequence based on previously seen data[21].

The model is defined by a set of states, each representing a nucleotide base (A, C, G, or T). The transition probability P_{ij} is the probability of transitioning from a state i to a state j.

In the simplest case, the Markov chain can be modeled as a first-order chain, meaning that the probability of transitioning from a state i to a state j depends only on the state i. By increasing the order, we get sequences that increasingly resemble the ones that the model was trained on.

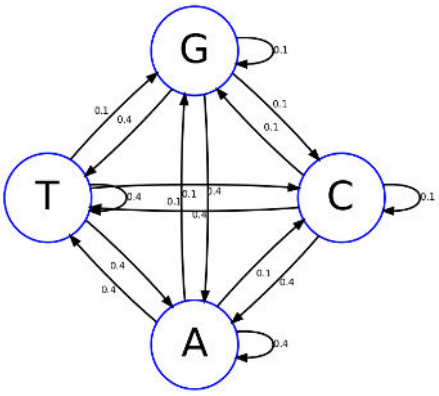

Figure 2.1: Example of a Markov chain[22]

After creating the MC, it is possible to generate a sequence of length n by sampling a nucleotide at each position based on the learned posterior probabilities. Using $k = 2$, sequences with 3-mers common to LTRs can be generated.

An implementation of this generative Markov chain model by K. Youens-Clark [21] was used to generate **15%** of sequences in the final set.

All of the previously mentioned sequences' lengths were sampled from a distribution of actual lengths of LTRs, to avoid length being a factor, as LTRs vary highly in size. In particular, the training database used consisted of lengths distributed in the following way:

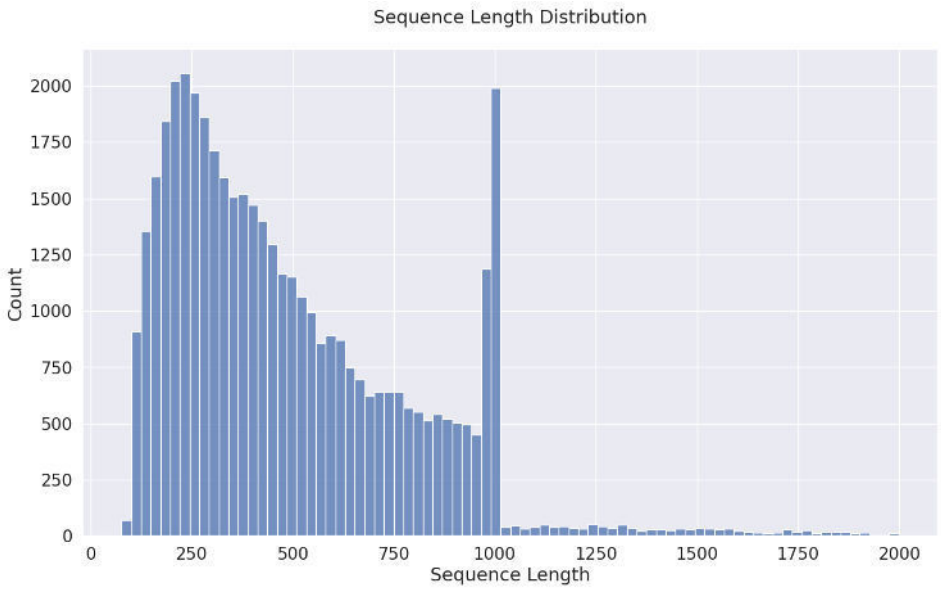

Figure 2.2: LTR Sequence length distribution, outlier sequences above 2000bps have been cut off

3 Transcription Factors and Information Content

As mentioned earlier, LTRs are specific regions that contain promoters, enhancers, and other regulatory sequences.

These specific DNA sequences serve as a recognition site for proteins involved in transcription. The promoter sequence is typically located upstream (toward the 5' end) of the transcription start site and is crucial in initiating gene expression. Such regions are known to contain specific DNA motifs, such as TATA and CAAT boxes.

Regulatory regions are also known to contain a plethora of so-called transcription factor binding sites. These sites are specific to their corresponding transcription factors, critical regulators of gene expression, and are involved in a wide range of biological processes, including development, differentiation, and response to environmental signals. Dysregulation of transcription factors can lead to various diseases, including cancer and autoimmune disorders[23]. Therefore, a large area of biological research is dedicated to analyzing the function of transcription factors and how they regulate gene expression. One such project is the JASPAR database.

3.0.1 JASPAR

JASPAR[24] is a curated, open-access database that contains a collection of transcription factor binding sites.

It is focused on "core" DNA binding motifs, which are short, conserved sequences recognizable by DNA-binding proteins. These motifs are typically 6-30 base pairs in length and are present in the regulatory regions of genes.

The database is organized into several collections, including the "core" collection containing the most widely studied and best-characterized motifs. Other collections include "phylogenetically-related" motifs, which are grouped based on evolutionary relationships, and "unvalidated" motifs, which are putative motifs that have not yet been experimentally validated.

Each motif in the JASPAR database is represented as a position-specific scoring matrix (PSSM), which captures the probabilities of

observing each of the four nucleotides at each position within the motif. The PSSMs are generated by aligning a set of known binding sites for a particular transcription factor and calculating the frequencies for each position. In addition to the PSSMs, the JASPAR database provides further information about each motif, such as its name, accession number, taxonomic range, and literature references.

3.1 PWMs and information content

A position weight matrix (PWM) is a commonly used representation of the consensus sequence of a DNA or RNA sequence region besides PSSMs. The PWMs used in this work were derived by first downloading PSSMs from the JASPAR database, loading them using the Bio.motifs module in the biopython[25] package, and then transforming these into corresponding PWMs.

PWMs may be highly specific to certain DNA sequence patterns or less specific (fit to a larger proportion of sequences). As we are working with nucleotide sequences, the expected probability $E(p)$ for each base at each position is $\frac{1}{4}$.

A PWM with the lowest possible specificity at a position has the following probability distribution:

$$(\mathbf{A:}\ \tfrac{1}{4}, \mathbf{C:}\ \tfrac{1}{4}, \mathbf{T:}\ \tfrac{1}{4}, \mathbf{G:}\ \tfrac{1}{4})$$

Whereas for a highly specific position in a PWM, it could look something like this:

$$(\mathbf{A:}\ \tfrac{9}{10}, \mathbf{C:}\ \tfrac{0}{10}, \mathbf{T:}\ \tfrac{0}{10}, \mathbf{G:}\ \tfrac{1}{10})$$

The JASPAR collection used contains numerous TFBS motifs that are less specific and tend to be matched to the query sequence more often than more specific ones. When conducting feature selection, or pattern mining, it is desirable to penalize such motifs more, as they tend to appear more often and could introduce a bias into the algorithm's results.

To solve this issue, it is possible to calculate the information content of such a matrix and use this value to weigh the support scores obtained during pattern mining or implement a weighted training process.

The total information content of a PWM[26] is calculated in the following way:

$$IC(PWM) = \frac{\sum_{i=1}^{|PWM|}(ME - I(PWM,i))}{log_2(|PWM|))}$$

$$I(PWM,i) = -\sum_{b}^{\{A,C,T,G\}} PWM[i][b] * log_2(PWM[i][b])$$

$$ME = log_2(4) = 2$$

$I(PWM,i)$ is the information content at position i in the PWM matrix, ME is the maximum entropy measure, and |PWM| is the length of the sequence that the PWM describes.

To illustrate, the following sequence logos represent motif PWMs with the highest IC and lowest IC, respectively.

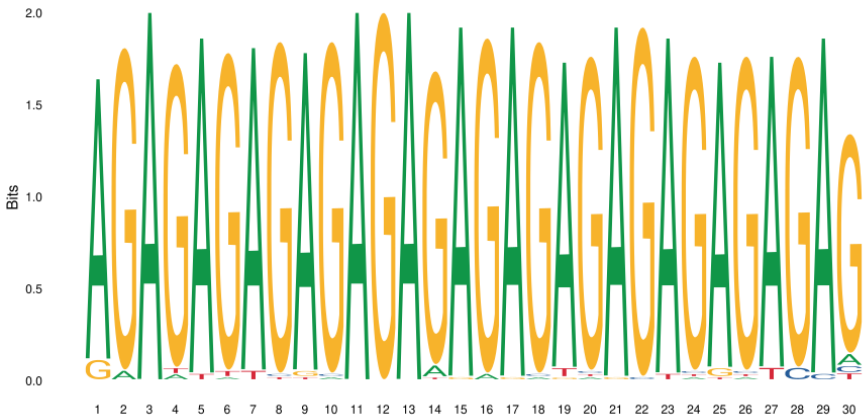

Figure 3.1: Highest IC motif: BPC5

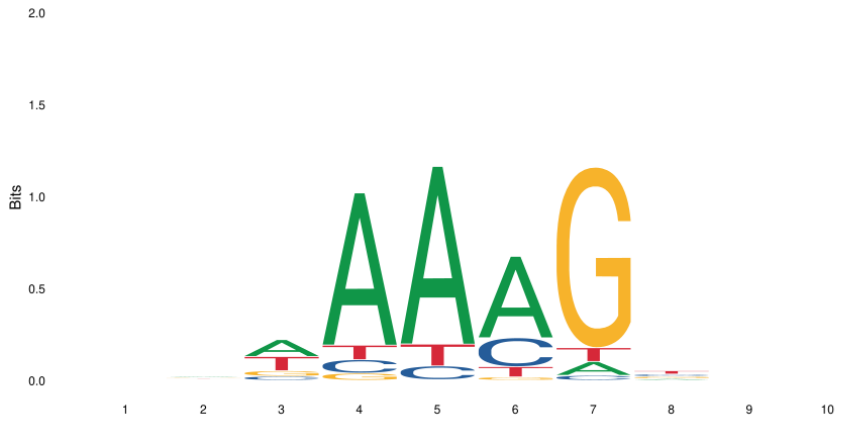

Figure 3.2: Lowest IC motif: DOF1.8

4 Frequent motif mining

Association rule mining is a technique for discovering significant co-occurrence among variables in large datasets. It has seen applications in various fields, such as marketing, bioinformatics, and web mining.

A crucial term in frequent pattern mining is **support**. This is the percentage of transactions or records in a dataset containing a particular item. In other words, it measures the frequency of occurrence of an itemset in the dataset. An itemset with high support is considered frequent, while an itemset with low support is considered infrequent[27].

$$Support(A) = \frac{N_A}{N}$$

In the context of LTR sequences, the detection of frequently co-occurring TF motifs could provide an insight into the functionality of LTRs, if motifs with a specific biological function were found to occur frequently together within these regions. For this purpose, the ECLAT algorithm was used.

4.1 ECLAT

Equivalence Class Clustering and Bottom-up Lattice Traversal or *ECLAT* is an efficient and scalable algorithm for mining frequent item sets in transactional databases. This chapter will provide an overview of the algorithm, its advantages and limitations, and how it was used in connection with the TFBS motifs found.

4.1.1 Introduction to ECLAT

The ECLAT algorithm was proposed by M.J. Zaki[28] as a faster and more memory-efficient alternative to the Apriori algorithm[27]. It uses a vertical data representation, storing transactions as columns rather than rows. This representation allows the algorithm to use bitwise operations for more efficient computation of the itemset support.

The algorithm consists of two main steps:

- **Equivalence class clustering**

- **Bottom-up lattice traversal**.

Equivalence Class Clustering

The first step of ECLAT is to cluster transactions into equivalence classes based on their shared items. Equivalence classes are sets of transactions that have the same set of items.

The algorithm then scans the database to compute the support of each item. An item is considered frequent if its support is above a minimum support threshold specified by the user. A tree-like structure called an itemset lattice is created, which represents all possible itemsets. The nodes represent the sets, and the edges represent the subset relationships between them.

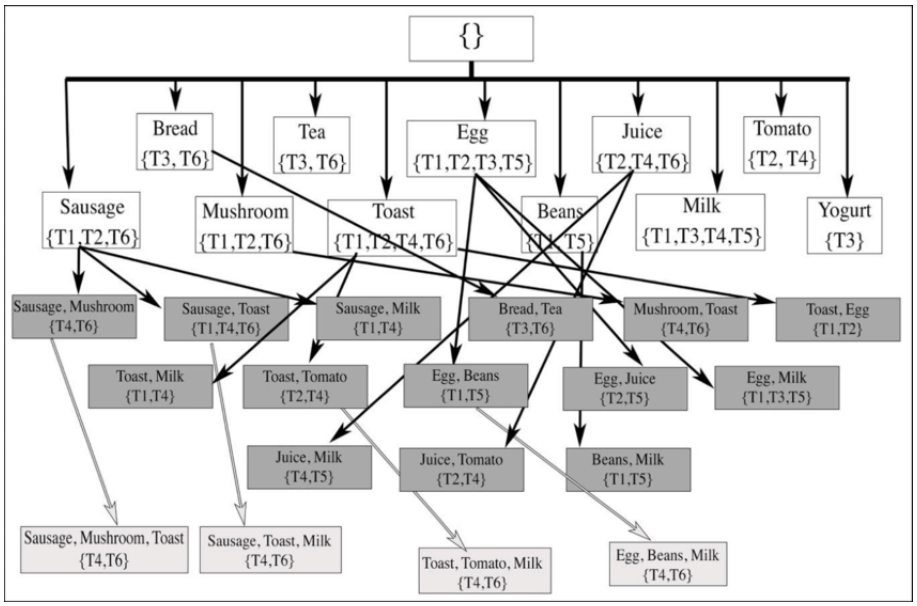

Figure 4.1: ECLAT inner tree representation[29]

Bottom-up Lattice Traversal

After clustering transactions into equivalence classes and constructing the itemset lattice, the ECLAT algorithm traverses the lattice in a bottom-up manner to discover all frequent itemsets. The traversal is performed using a depth-first search algorithm.

At each node, the algorithm checks if the itemset is frequent and adds it to the frequent list. Its immediate children are then recursively visited. These children can be thought of as the itemsets that can be obtained by extending the current itemset by a single item.

4.1.2 Advantages and Limitations

The ECLAT algorithm has several advantages over the Apriori algorithm[30]:

- **Memory Requirements:** The vertical data representation is more memory-efficient than the horizontal data representation used by Apriori
- **Number of Computations:** The ECLAT algorithm does not involve the repeated scanning of the data to compute the individual support values.
- **Speed:** The DFS approach makes the ECLAT algorithm faster than the Apriori algorithm in the average case.

One of the algorithm's limitations is that it requires the database to fit into the main memory, which may be a limiting factor for large datasets. Another limitation is that the ECLAT algorithm may produce a large number of candidate itemsets, which can be time-consuming to process during run time.

Both of these problems were encountered during the runs I conducted using the algorithm. The frequent itemsets of size two were reached relatively quickly (a matter of minutes), even for a large number of motifs (initially 638 motifs and 44029 sequence entries); however, while analyzing the itemsets of size 3, the algorithm's time complexity grew exponentially, and exceeded run times of over four days, before being killed prematurely.

A smaller number of candidate motifs had to be used to reduce the time and memory requirements. The **information content criterium** was used to select 100 motifs with the highest IC value, on which the algorithm was run.

N motifs	Size 2 time (s)	Size 3 time	Size 4 time
635	∼23m	>4d	N/A
100	34s	∼9m	∼160m

Table 4.1: ECLAT algorithm runtimes

4.1.3 Observations

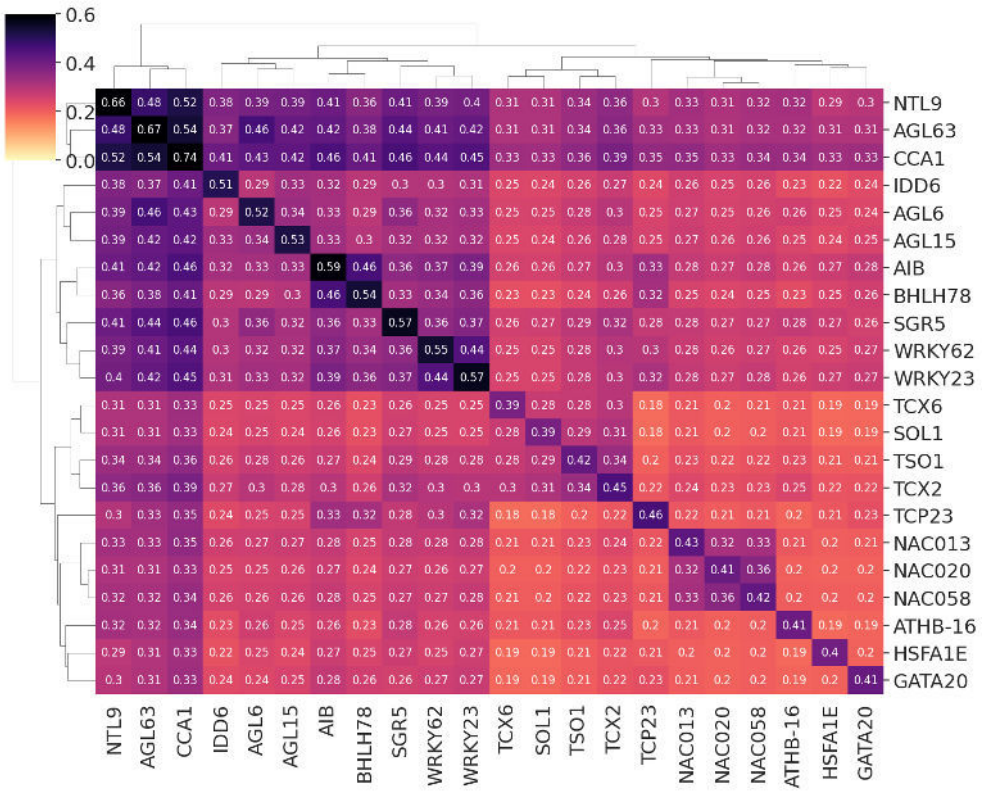

Figure 4.2: Clustermap of TF motif pairs with highest support scores. Clusters of motif groups are visible. These represent motifs that share similar support values within subsets of the top co-occuring observations.

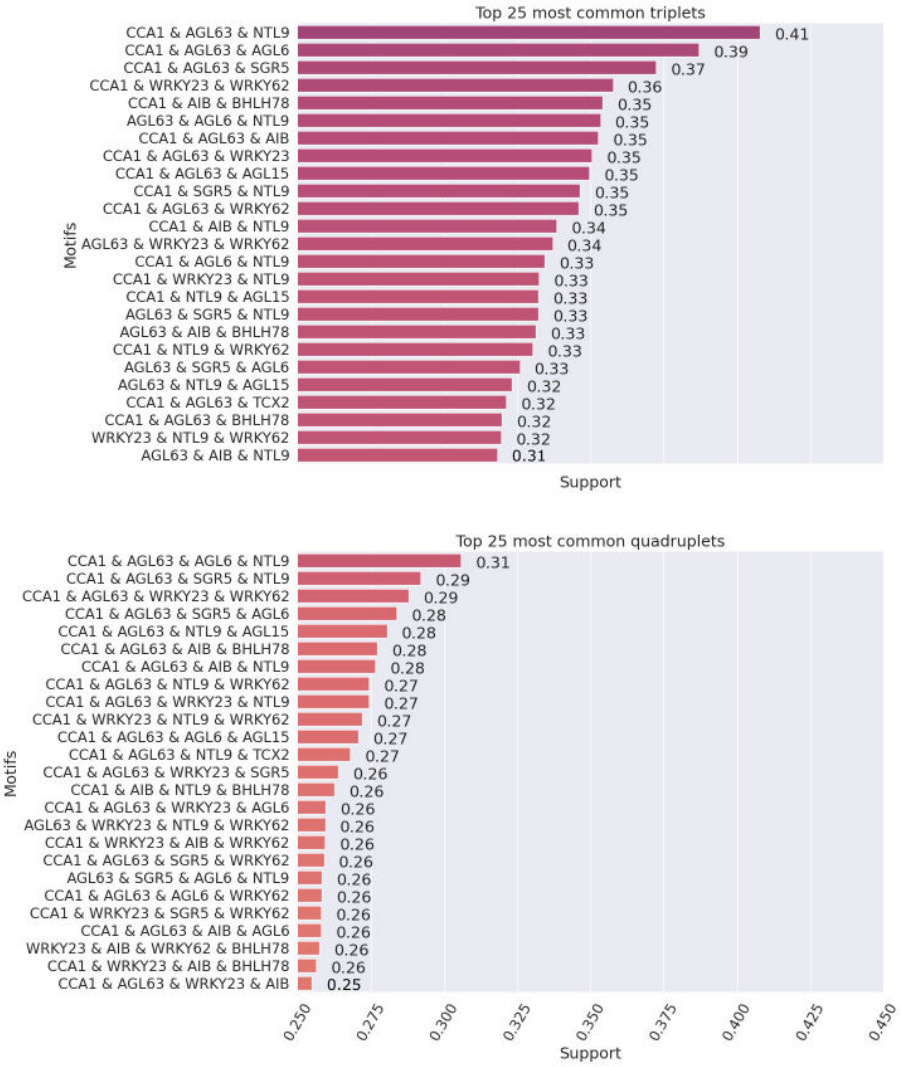

Figure 4.3: Itemsets of size 3 and 4 with the highest support scores.

Name	Function
CCA1	Transcription factor involved in the circadian clock and in the phytochrome regulation[31]
AGL63	Probable transcription factor involved in the regulation of fruit growth. Contributes to integument development [32]
NTL9	Mediates osmotic stress signaling in leaf senescence by up-regulating senescence-associated genes [33]
AGL15	Transcription factor involved in the negative regulation of flowering[34]
SGR5	Regulates lateral organ morphogenesis and gravitropic responses [35]
WRKY23	Transcription factor involved in mediating auxin feedback on PIN Polar Localization[36]
AIB	Transcription activator. Regulates positively abscisic acid (ABA) response. Confers drought tolerance and sensitivity to ABA [37]
BHLH78	Binds to chromatin DNA of the FT gene and promotes its expression, and thus triggers flowering in response to blue light [38]

Table 4.2: Top co-occurring motifs with descriptions from UniProt[39]

Common tuples were statistically tested using **g:Profiler**[40]. The *g:GOSt* tool of g:Profiler performs functional enrichment analysis of the input genes, mapping them to known functional sources and extracting statistically significant terms. This can be useful in detecting common biological functions within the query list.

At a significance level of 0.05, the only statistically significant function detected is the negative regulation of circadian rhythm (CCA1) and auxin polar transport (WRKY23 and SGR5).

Looking at the top motifs co-occurring in itemsets, the most representative ones are involved in various biological processes, such as the mentioned circadian clock rhythm, fruit growth, flowering, and different environmental responses such as stress and gravitropism.

These broadly include signaling pathways and could mean plants have adapted LTRs as signaling pathway regulators.

5 Simple Model Training

For the task of classification of LTRs, JASPAR motif occurrences are used as training features for the models, as the variable length of these sequences hinders their use in conventional ML models without preceding preprocessing steps. Each sequence was parsed for each motif, and a vector $(x_1, ..., x_{638})$ was produced, where x_i represents the number of occurrences of motif i in that particular DNA sequence.

5.1 Feature transformation using TF-IDF

The *TF-IDF* algorithm has been used in all pipelines to scale and weigh the data.

TF-IDF stands for Term Frequency-Inverse Document Frequency. It is a numerical statistic commonly used in natural language processing and information retrieval to evaluate the importance of a term in a corpus of documents.[41]

The TF-IDF algorithm assigns a weight to each term based on how often it appears in that particular document and how rare it is across the entire corpus. The algorithm consists of two main components:

Term Frequency (**TF**): This metric measures the frequency of a term in a document, usually by simply counting the number of times the term appears in the document. The more often a term appears in a document, the more important it is to that document.

$$TF(t, D) = \frac{|D_t|}{|D|}$$

where $|D_t|$ is the number of times the term t appears in document D, and |D| is the total number of terms in document D.

Inverse Document Frequency (**IDF**): This component measures the rarity of a term across the entire corpus. Rare terms that appear in only a few documents will have a higher IDF score, while common terms that appear in many documents will have a lower IDF score.

$$IDF(t, C) = log_2(\frac{|C|}{|C_t|})$$

Then **TF-IDF** is calculated as:

$$TFIDF(t, D) = TF(t, D) * IDF(t, D)$$

This gives more weight to terms that appear in a document frequently but rarely in the corpus overall, indicating that they are more important or relevant to that document.

In the case of motifs, the most common receive lower TF-IDF values, shifting the focus toward the less common, sequence-specific ones. In the context of the motif feature weighing, this works similarly to weighing by the information content measure.

5.2 Classifier

Three Classifiers were tested in this task:

- **Multilayer Perceptron** - robust, generalizes well, and can handle classification tasks that are not linearly separable. These models are, however, prone to overfitting and are hard to interpret.

- **Gradient Boosting classifier** - Handles outliers well, highly accurate, however sensitive to hyperparameters and prone to overfitting on noise.

- **Random Forest Classifier** - built-in feature selection, generalizes well but may struggle with high-dimensional data.

5.2.1 Pipeline

In order to chain the preprocessing and classification steps, the Pipeline approach provided by the scikit-learn libary[42] was implemented. Pipelines are an efficient way of condensing all the preprocessing steps into one object, making it easier to work with. The transforms are sequentially applied to the input data, where each input of a transformer object within the pipeline is the output of the previous.

This approach enables more concise code and easier deployment of trained models.

5.2.2 Grid Search tuning

Grid search is a hyperparameter tuning technique used in machine learning to find the optimal values for a given model. It involves selecting a set of values for each hyperparameter of a model and then systematically evaluating the model's performance for that combination using stratified k-fold cross-validation and a performance metric such as accuracy or F1-score.

The grid search results are displayed in the following parallel coordinates, where vertical axes represent a value range for the tested hyperparameter, and the last axis contains the k-fold cross-validation accuracy score.

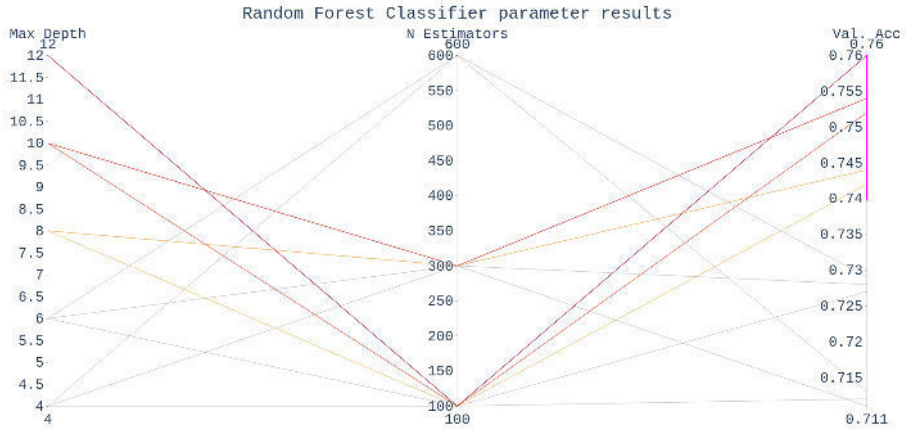

Figure 5.1: RFC grid search results. Higher values of the *Maximum depth* parameter and lower values of the *number of estimators* used yield the best results in this case. This is likely due to better generalization properties of the classifier when fewer trees are used to build the ensemble model.

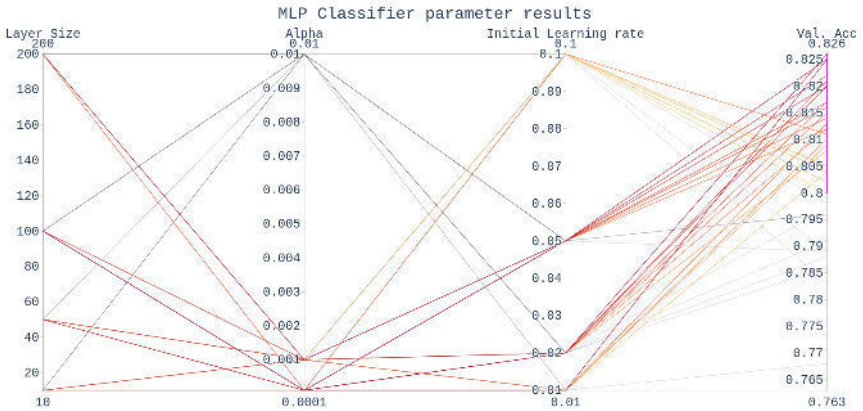

Figure 5.2: MLPC grid search results. *Layer size* and *initial learning rate* do not seem to play a significant role in the model performance. Lower *alpha* values, on the other hand, lead to much better results in the model.

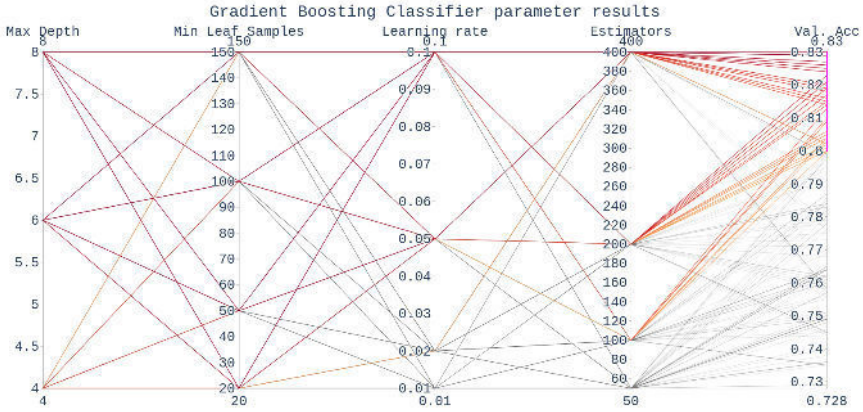

Figure 5.3: GBC grid search results. Here, the hyperparameters have less influence on the model performance than in the previous two cases. There is, however, a visible trend for the *learning rate* where higher values seem to have contributed to faster convergence of the model and *estimators* parameter where higher values produce a more sensitive model.

Classifier	Parameter Values	Accuracy %
RFC	Max Depth: 12, N Estimators: 100	76%
MLP	Layer Size: 100, α: 0.0001, Learning Rate: 0.02	82.6%
GBC	Max Depth: 8, Min. Leaf Samples: 150, Learning Rate: 0.1, Estimators: 400	**83%**

Table 5.1: Scores of best-achieving parameters from each tested classifier for JASPAR features. The best performance was achieved by the **Gradient Boosting classifier** with a cross-validation accuracy of 83%.

5.3 Feature analysis and Importance

The frequent patterns seen before may indicate features commonly occurring within data, however, do not always represent features that contribute to classification in a significant way. In order to detect highly influential TF motifs, the GPUTreeExplainer from the SHAP[43] package was used on the trained optimal RFC model. The GPUTreeExplainer works by constructing a set of decision trees that are representative of the original model. The trees are then used to estimate the expected Shapley values for each feature in the input dataset. Shapley values are a measure of the average contribution of that attribute to the model output across all possible attribute combinations.

Its advantage over the basic TreeExplainer is built-in GPU support, making the original, computationally expensive algorithm run much faster.

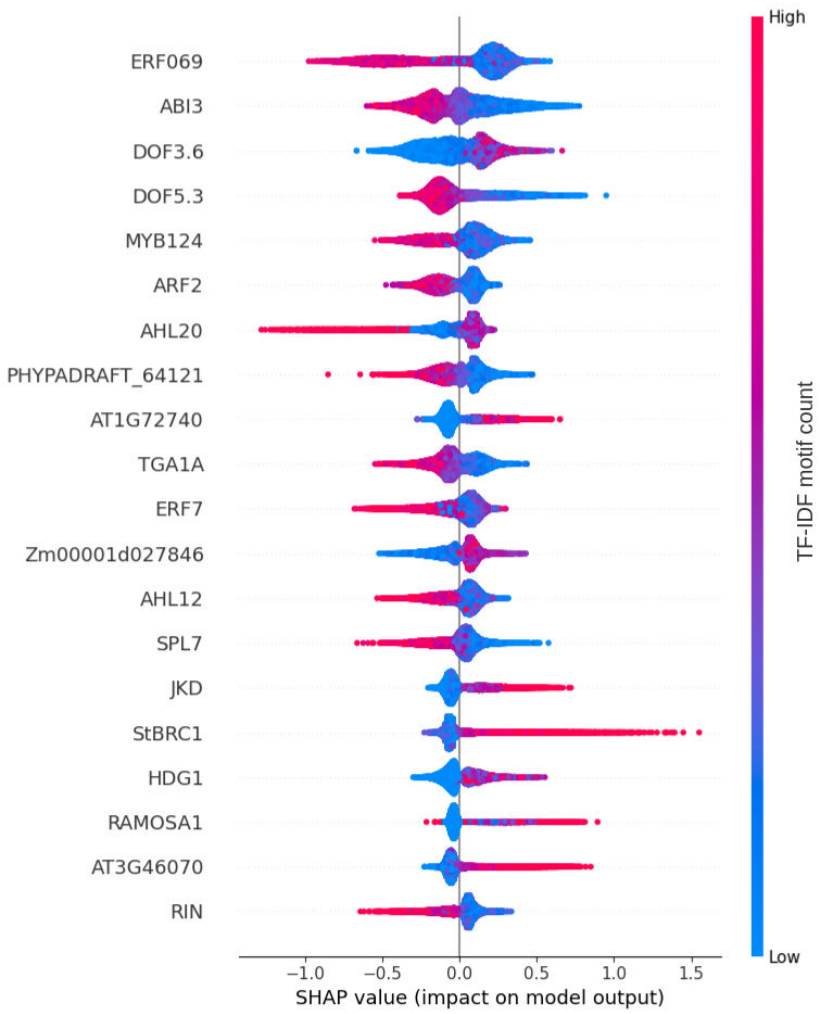

Figure 5.4: In the summary plot above, we can see the top 20 influential motifs. The vertical line in the center indicates the mean SHAP value across all input features, and points on the horizontal axis represent separate occurrences of the motif in a sequence. The *TF-IDF motif count* represents the number of occurrences of a motif within a sequence transformed by TFIDF. This allows us to see that motifs with a higher TFIDF value (red) spanning to the left side contribute to negative classification (non-LTR). In contrast, those spanning towards the right side contribute to positive classification (LTR). We may conclude that motifs such as **JKD**, **StBRC1**, and **AT3G46070** can be considered more specific to LTR sequences than, for example, the **ERF069**, **AHL20**, or **RIN**, whose higher values contribute to the classification negatively.

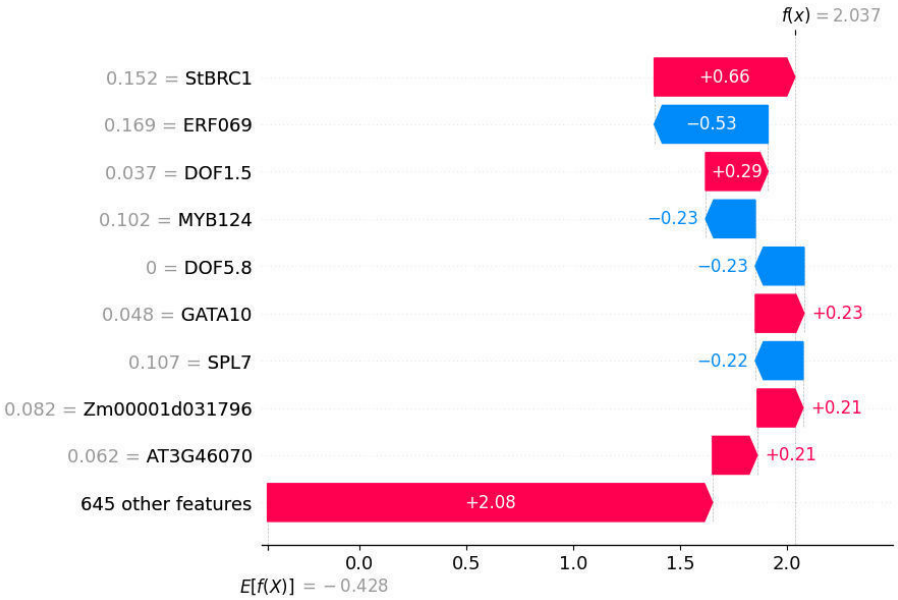

Figure 5.5: The waterfall plot is another way to visualize the results of the SHAP algorithm. Each bar in the plot represents the contribution of a single feature to the prediction. Bars are colored according to the direction of the feature's effect, red: positive, blue: negative. Here we confirm the contribution of the **StBRC1** and **AT3G46070** motifs as significant for LTRs, and three other significant motifs emerge: **DOF1.5**, **GATA10**, and **Zm00001d031796**

5.3.1 The biological function of influential motifs

Name	Function
StBRC1	May be involved in Aerial and Underground Lateral Branching [44]
DOF1.5	Acts as a negative regulator in the phytochrome-mediated light responses [45]
GATA10	May be involved in the regulation of some light-responsive genes
Zm00001d031796	participates in the plant hormone transduction pathway [46]
AT3G46070	Response to chitin stimuli, and regulation of DNA-templated transcription[47]
JKD	Controls position-dependent signals that regulate epidermal-cell-type patterning [48]

Table 5.2: SHAP detected influential motifs

From the results of g:Profiler, the JKD motif is significantly represented in epidermal development and cell differentiation and could, therefore, also be involved in leaf and flower growth.

The biological function of the DOF1.5 and GATA10 motifs is linked to light responses. In the case of Zm00001d031796, the connection to ethylene-activated pathways may indicate the TF's relation to stress responses, as ethylene has been linked to stress-induced processes in plants and flowering pathways[49].

6 Neural Network classification

The TF motif preprocessing approach offers a relatively fast and simple way to preprocess variable length sequences into fixed-size feature vectors but fails to capture positional relationships within the observed sequence. This could impact the recognition of LTRs, as the functionality of regulatory regions can depend not only on the presence of sequential motifs but also on their positional context.

This chapter describes a classification training workflow using Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM), which have recently seen a wide range of uses in the area of DNA sequences classification tasks such as metagenomics analysis[50] and viral sequence detection [51].

6.1 Sequence Encoding

A simple one-hot encoding approach was used to represent the input sequences, where the bases were represented in the following way:

A:	[1, 0, 0, 0]
T:	[0, 1, 0, 0]
C:	[0, 0, 1, 0]
G:	[0, 0, 0, 1]
N:	[0, 0 ,0 ,0]

Table 6.1: Sequence encoding

In this encoding scheme, each nucleotide is represented by a binary vector of length 4, where the position of 1 in the vector signifies the presence of the corresponding base.

6.2 Sequence Padding

When training models, it is generally easier to work with fixed-length inputs, as this allows the use of matrix operations and efficient parallelization techniques. Padding and truncation are two major ways to ensure a sequence fits a predetermined length.

Since sequence truncation could potentially lead to loss of information if an important region is cut off, padding was chosen as the go-to technique for handling this problem. Truncation is only used to shorten outlier sequences above 2000bps.

Three groups of sequence lengths were created: [0-350), [350-700), and [700-2000). Within each group, sequences shorter than the maximum length are padded by appending additional values (the vector [0,0,0,0] in this case) to their end until they fit the predetermined size.

These groups were chosen in a way that maintained a sensible length distribution between the set of sequences and contained enough instances for each of the three models to be trained on.

6.3 Network Architecture

6.3.1 Convolutional neural network (CNN)

CNNs use filters to detect local patterns in the input sequence to pool together neighboring segments. In the case of DNA sequence classification, this can be used to effectively capture important patterns and features with a significant biological function, such as domains[52], codons, protein binding sites[53] or nucleosome positioning[54].

CNN filters can identify these patterns in the input sequence, regardless of their exact location, making them particularly effective at recognizing these in large and complex DNA sequences.

Additionally, CNN layers can automatically learn and adapt to different features in the data, making them highly effective at generalizing to new and unseen inputs. This is an important property in classification tasks. The goal is often to accurately classify new sequences not used in the training set, which may be highly divergent.

The CNN filter size is considered to be directly related to the size of the recognizable motif that the network can detect [55], as it marks the width of the region of the input sequence that the filter scans at a time. If the filter size is smaller than the motif size, the filter may detect only parts of the motif and could miss the entire pattern. On the other hand, if the filter size is much larger than the motif size, the filter may pool the motif with other, unrelated patterns.

Therefore, choosing the appropriate filter size is important based on the size of the motifs being searched for. Generally, the filter size is

chosen to be slightly larger than the motif size to ensure the filter can capture the entire segment.

For example, if the expected motif size is 10, a filter size of 12 or 14 nucleotides may be appropriate. The exact filter size is often empirically determined through experimentation and optimization or from domain-specific knowledge.

6.3.2 Long Short term memory (LSTM) nodes

An LSTM is a recurrent neural network layer commonly used in deep learning models to process sequential data such as speech, text, or time series.

The purpose of an LSTM layer is to allow a neural network to remember important information from previous time steps or sequences and to selectively forget less important information. This is achieved through the use of a series of gates that control the flow of information through the layer.

An LSTM layer typically consists of three gates:

- **Forget gate**: Determines which information from the previous hidden state and the new data point in the sequence should be discarded. It helps the network discard irrelevant information.
- **Input gate**: Determines which new information from the current time step should be added to long-term memory.
- **Output gate**: Determines which information from the current hidden cell state should be used to generate the output for the current time step.[56].

LSTM layers may be suitable for DNA classification in combination with other layer types thanks to their ability to capture long-term dependencies. This enables the network to connect motifs of the input sequence located at different regions along its length.

Besides this, LSTMs can handle noise within the input (using the forget gate). This property comes in handy, as it is possible that large regions of LTR sequences could hold little to no regulatory value and should therefore have less impact on the classification of these sequences.

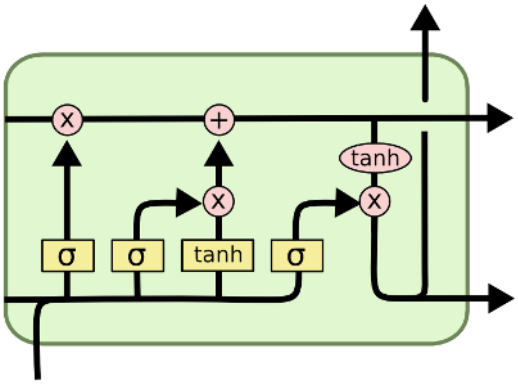

Figure 6.1: An LSTM node[57]

By combining CNN and LSTM layers, the network can take advantage of the strengths of both types of layers. The CNN layers can extract useful features using its fixed-size filters from the input data, which the LSTM layers can then use to draw predictions from.

In practice, the combination has shown promise in text classification tasks, being able to capture both significant areas and their long-term context[58].

6.4 Comparing architecture setups

Four tests were conducted for each of the three length categories in order to detect the most viable combination of the mentioned NN node types.

The dataset was split into 60% training sequences, 20% validation sequences, and 20% testing sequences.

The four tested architectures were the following:

- 3 Convolutional layers connected by max-pooling (**LARGE**)
- 3 Convolutional layers connected by max-pooling (**SMALL**)
- 3 Convolutional layers connected by max pooling and 1 LSTM layer

- 2 Convolutional layers connected by max pooling and 1 LSTM layer

Each was trained using the Adam optimizer, with binary cross entropy as a loss function, for 25 epochs, using the early stopping callback to prevent overfitting the model.

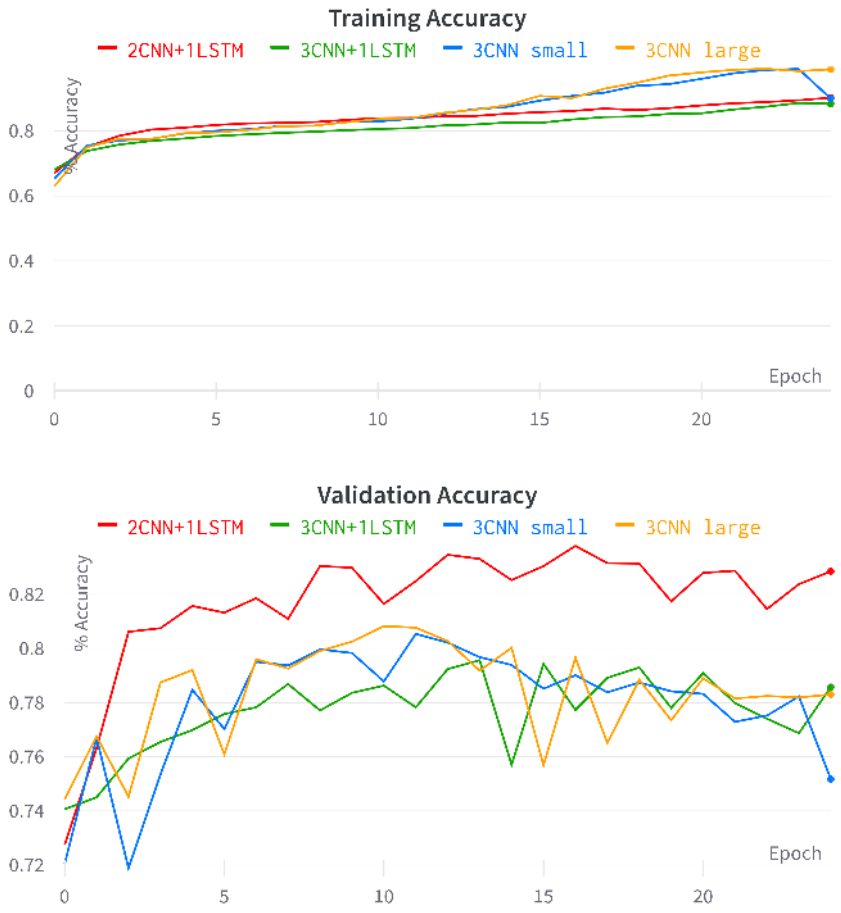

Figure 6.2: [0,350) group training and validation accuracies. The 2CNN+1LSTM architecture is used in the final hyper-tuned model

Figure 6.3: [350,700) group training and validation accuracies. The 2CNN+1LSTM architecture is used in the final hyper-tuned model

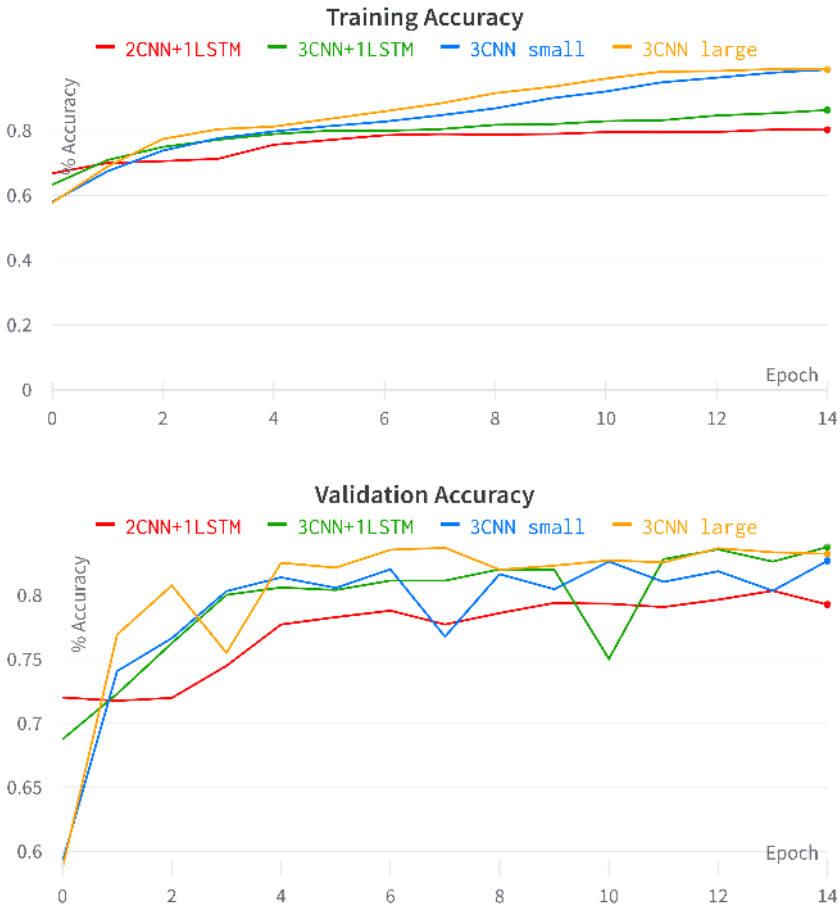

Figure 6.4: [700, ∞) group training and validation accuracies. The 3CNN+1LSTM architecture is used in the final hyper-tuned model.

6.5 Hyperparameter tuning using Keras tuner

Keras Tuner[59] is a Python module that provides an easy and efficient way to tune hyperparameters for deep learning models built with Keras[60].

Analogous to the GridSearch method in scikit-learn, Keras Tuner can be used to search the hyperparameter space for optimal settings by training and evaluating a series of models with different config-

urations. The library supports several search algorithms, including random search, grid search, and bayesian optimization.

The hyperparameter tuning is done by first defining the model architecture and then creating a list of hyperparameters (filter size, kernel size, or number of LSTM nodes) to be tested.

Besides architectural hyperparameters, multiple training arguments may be tested, such as learning rate, optimizer to be used, as well as the search algorithm that the tuner should implement.

The tuning process was linked to the Weights & Biases runtime interface [61], which provides easy experiment tracking, callbacks, and system resource monitoring.

6.5.1 Fine-Tuning observations

As seen in the following parameter importance scheme, the kernel size in the third convolutional layer is the most correlated with validation accuracy. Another observation is that the validation grows with a higher number of filters in the first layer and a lower number of dense units.

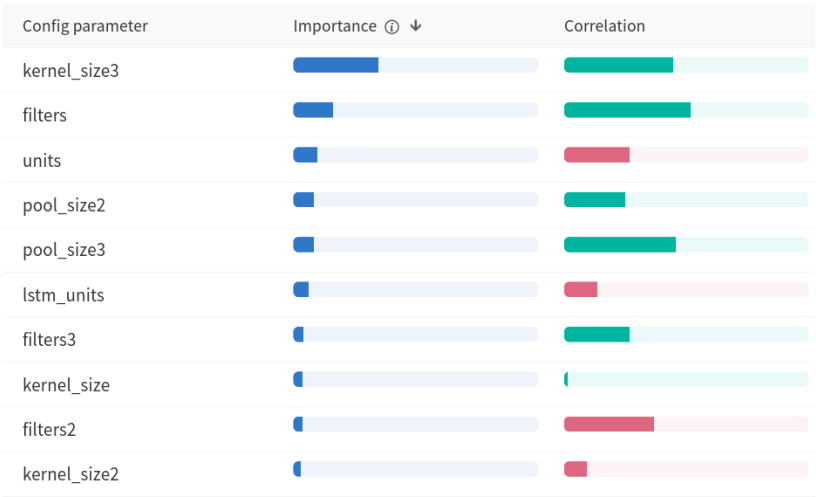

Figure 6.5: Observed correlation of hyperparameters and validation accuracy

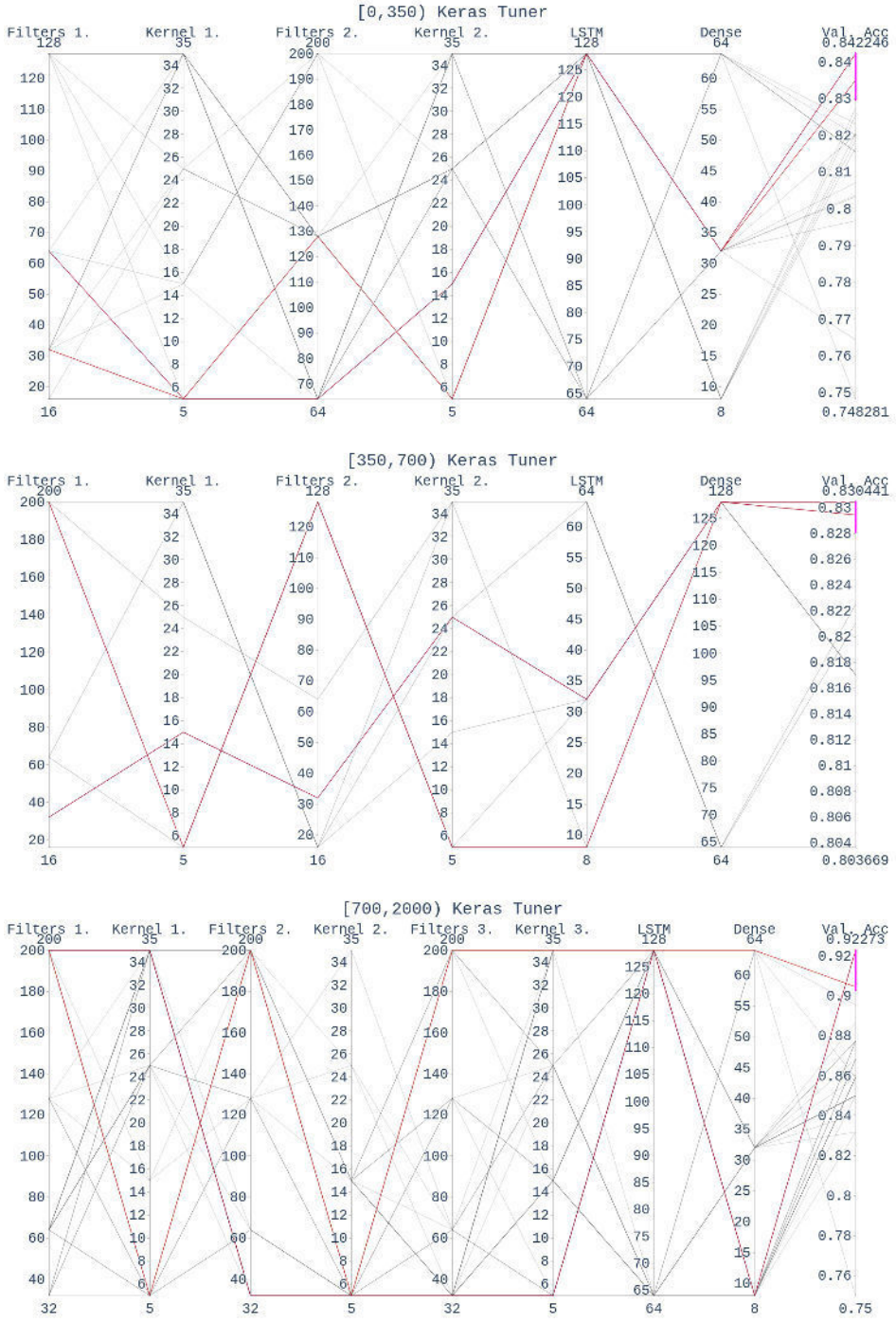

Figure 6.6: KerasTuner best estimated parameters

7 Bidirectional Encoder Representations from Transformers (BERT)

BERT [62] is a deep neural network model based on the Transformer architecture. The Transformer is a self-attention mechanism that processes input data in parallel, allowing it to handle long sequences of text efficiently. BERT consists of a multi-layer bidirectional Transformer encoder, where each layer contains multiple self-attention heads and fully connected feed-forward networks.

7.1 Advantages and disadvantages of BERT

BERT is a bidirectional model, which means it can comprehend text from both directions. This allows it to capture the meaning of a word based on the words that come before and after it in a sentence, as well as the overall context. Other than that, it can handle long-term dependencies within the input text.

Popular pre-trained versions of the model, such as BERT-Base[62] (pre-trained on a large corpus of text from Wikipedia), and BioBERT[63] (pre-trained on a corpus of biomedical texts), exist, which set a baseline for more specific fine-tuning tasks. The pre-training is an unsupervised process and involves two main steps:

- **Masked language modeling** - this involves randomly masking out words in a sentence and training the model to predict the missing words based on the context of the sentence.
- **Next sentence prediction** - this involves training the model to predict whether two sentences are likely to appear next to each other in a given text sequence.

Once the model has been pre-trained, it can be fine-tuned on a particular NLP task, such as sentiment analysis or question answering in a specific domain. The pre-trained property means the model can fit faster and requires less training data.

A few downsides include the high computational complexity and strong dependence on the quality of the fine-tuning data[64].

7.2 DNA_BERT

In the context of DNA sequence classification tasks, BERT can be used to create embeddings of DNA sequences that can capture their inner structure. Like natural language, DNA sequences are composed of units arranged in a specific order to create meaning (encoding a particular protein, TFBS motifs). It has been found that approaches applicable in NLP can also be applied in DNA sequence classification [65].

The advantage of using BERT for DNA sequence classification tasks is that it can learn complex features of the data without requiring hand-engineered features or domain-specific knowledge. This is particularly useful, for example, in genomics, where there is a vast amount of data but a limited understanding of the biological mechanisms that underlie it. BERT can learn from the patterns in the data to create useful representations that can be used for downstream tasks, such as predicting the effects of genetic variants or classifying sequences based on their function.

DNA_BERT[66] is a pre-trained model based on the BERT architecture specifically trained on DNA sequences, using kmer of size 6 as input tokens.

In this work, two versions of DNA_BERT were fine-tuned on LTR sequences. Each group consisted of a range of sequence lengths. One was trained on sequences within the length range [0, 350)bps and the other on sequences [350, 512)bps, 512 being the maximum input sequence size of traditional BERT models. Target site duplications were removed from the training sequences, as these are common among LTRs and could contribute to overfitting.

The models were then fine-tuned using the Adam optimizer, with an initial learning rate value of $1e^{-5}$, for 4 epochs. Both reach their peak validation accuracy around epoch 3 (88,5% and 86% respectively). This is rather fast for a model of this scale and is one of the advantages mentioned earlier in this chapter.

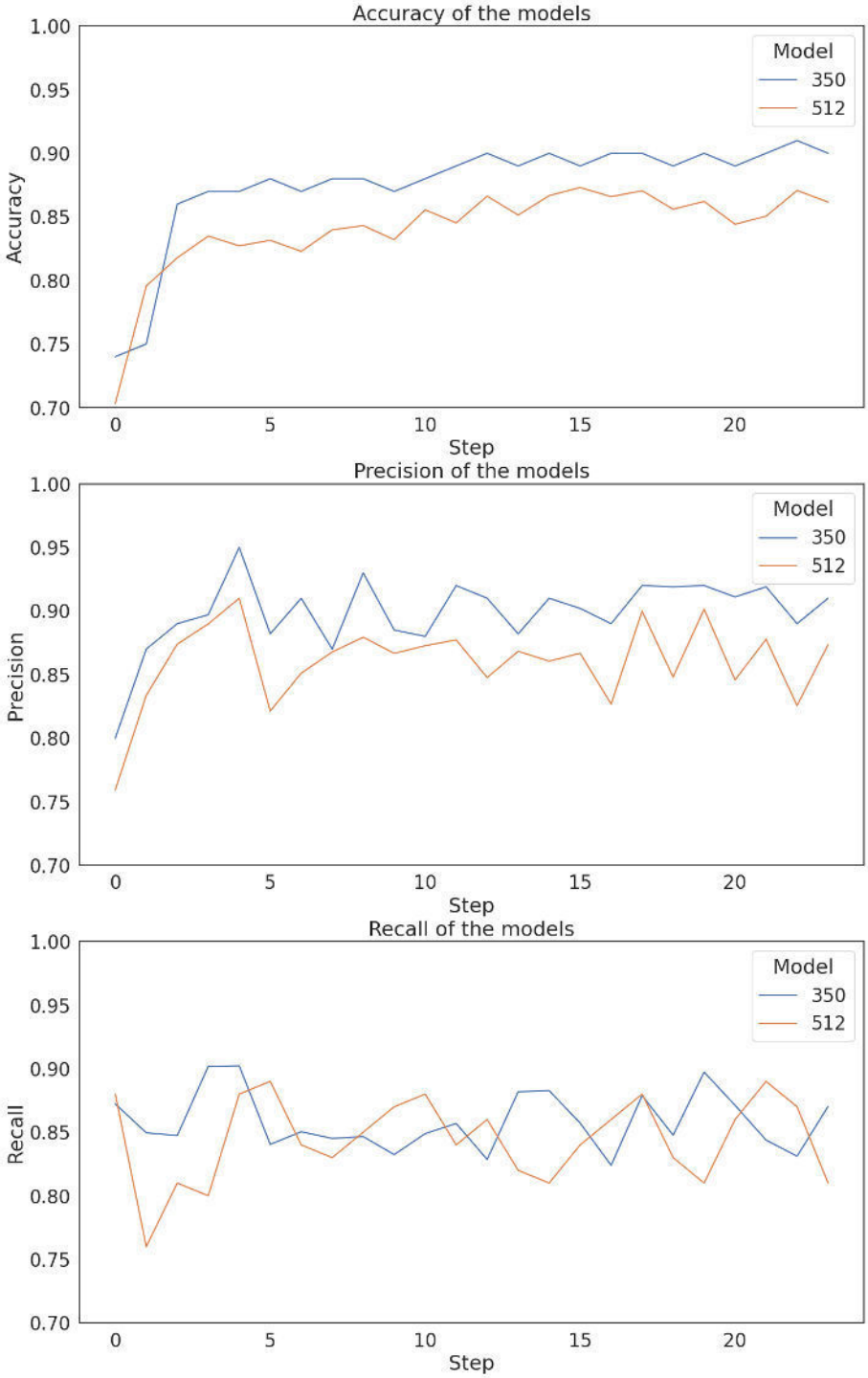

Figure 7.1: BERT model training performance

7.2.1 Large sequence classification

For sequences longer than 350 and 512, a max pooling technique was used to pool the embeddings of the DNA_BERT encoder and parse large sequences using a sliding window. The window size corresponds to the input size of the encoder (350 model used in this case). It is shifted by 1/3 of the model's max input size, meaning that with each movement along the sequence, 2/3 of the previous segment are maintained in the input, and the last 1/3 that is appended belongs to the next part of the sequence. In practice, this means that the window of size 350 is moved by 116 bases each time. The outputs of the BERT model on each such segment are then pooled by a convolutional network layer with a filter of size 3, meaning that three consecutive segments are being pooled together to avoid losing information from neighboring regions.

The following pseudocode shows how this approach works:

Algorithm 1: BERT-CNN pooling

```
sequence ← InputSequence
inp_size ← 350
model ← TrainedBertModel
index ← 0
embeddings ← []
while |sequence| ≥ index − inp_size do
    segment ← sequence[index : index + inp_size]
    embedding ← model.predict(segment)
    embeddings.append(embedding)
    index ← index + inp_size/3
end while
CNN ← TrainedNetwork
class ← CNN.predict(embeddings)
```

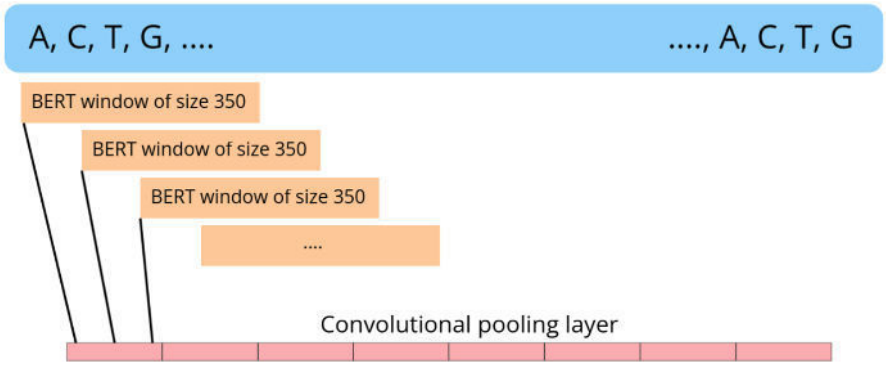

Figure 7.2: BERT-CNN sequence processing

7.3 Interpreting BERT results

7.3.1 Attention scores

Attention scores in BERT refer to the weight importance assigned to each token in a sequence by the attention mechanism of the transformer architecture. The attention mechanism is a key component of the transformer model used in BERT, allowing the model to focus on different parts of the input sequence while processing it.

During the attention process, BERT computes a set of attention scores for each token in the input sequence by comparing it to all the other tokens. These attention scores have been used to weigh each token's contribution to the sequence's final representation.

Here, I have taken a sample of 1000 LTR sequences under 350bps, averaged their attention scores using a window of size 20 for neighboring positions, and clustered them using an **Agglomerative Clustering** algorithm into five groups in order to group sequences with similar important positions together and visually enhance the important attention regions.

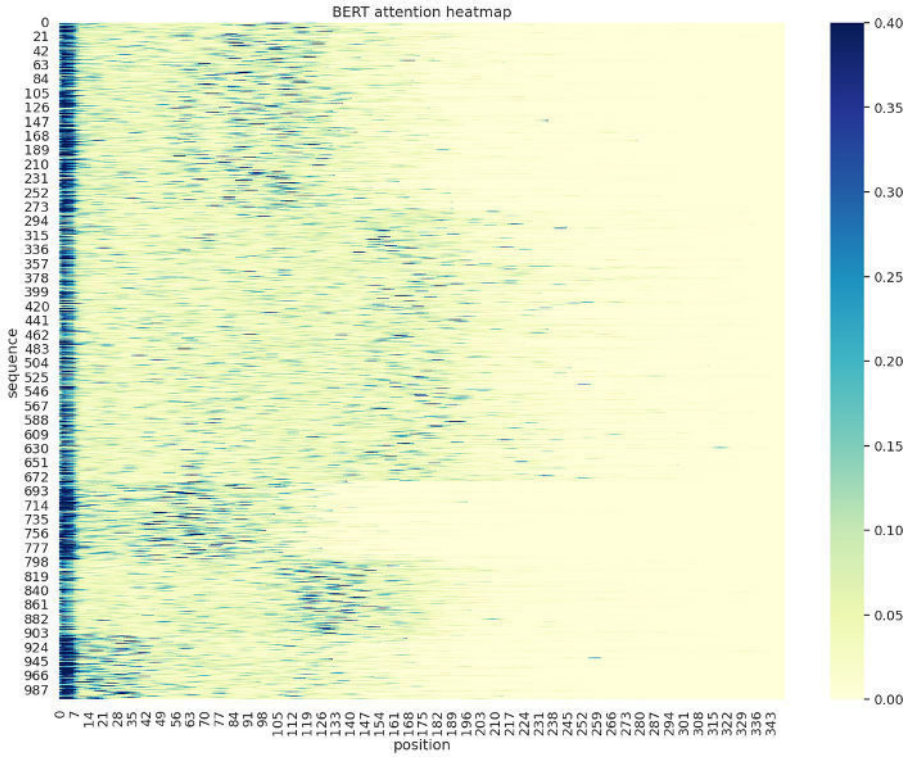

Figure 7.3: BERT attention scores for sequences <350bps clustered into similar groups, using 0 padding at the end of the sequence to fit into 350bps

Generally, the most distinct attention scores are located at the beginning and toward the end of LTR sequences. This is apparent for four of the five groups that emerge from the clustering and signifies that 6-mers highly specific to LTRs are located in these regions. The fifth group differs in that its highest attention scores are scattered around the beginning of the sequences. These could include LTRs whose influential regions are located at the 5′ of their sequence and could signify that these sequences contain regulatory motifs further upstream.

7.3.2 SHAP

The SHAP[43] has already been mentioned in this work in connection with the TreeExplainer 5.4.

Similar to the TreeExplainer, the SHAP method applicable for BERT works by generating a set of explanations for each prediction made by the model. Here, the SHAP algorithm generates a modified input sequence by replacing tokens with a special [MASK] token. The model is then run on each modified input, and the difference in output prediction is used to calculate the SHAP value.

Single sequence analysis

The Python SHAP package provides tools for the interactive visualization of values calculated by this algorithm.

One such tool is the Text plot[67] which can be used to view parts of the input text that contribute to the classification the most.

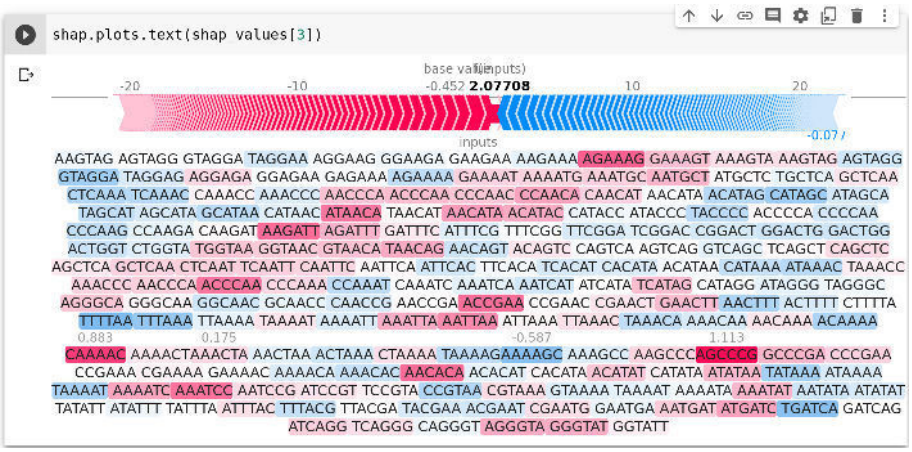

Figure 7.4: BERT SHAP text plot

This plot can be interacted with, and each "word", in this case a DNA sequence k-mer, can be viewed separately to analyze its contribution, with red color meaning positive contribution, i.e., towards classifying the element as an LTR, and blue meaning negative contribution or classifying the element as a non-LTR.

An interactive notebook has been created for this task, where the fine-tuned DNA_BERT model is loaded from the HuggingFace[68] repository, and the user may load their custom set of LTR-sequences to analyze.

Clustered analysis

The model fine-tuned on sequences under 350bps was analyzed by applying the SHAP algorithm to a sample of 300 LTR sequences. For this larger-scale analysis of the important sequence segments, an approach similar to that of visualizing BERT attention scores was taken, where sequences were clustered into five similarity groups using the previously mentioned agglomerative clustering algorithm, and clusters were plotted together on the following heatmap.

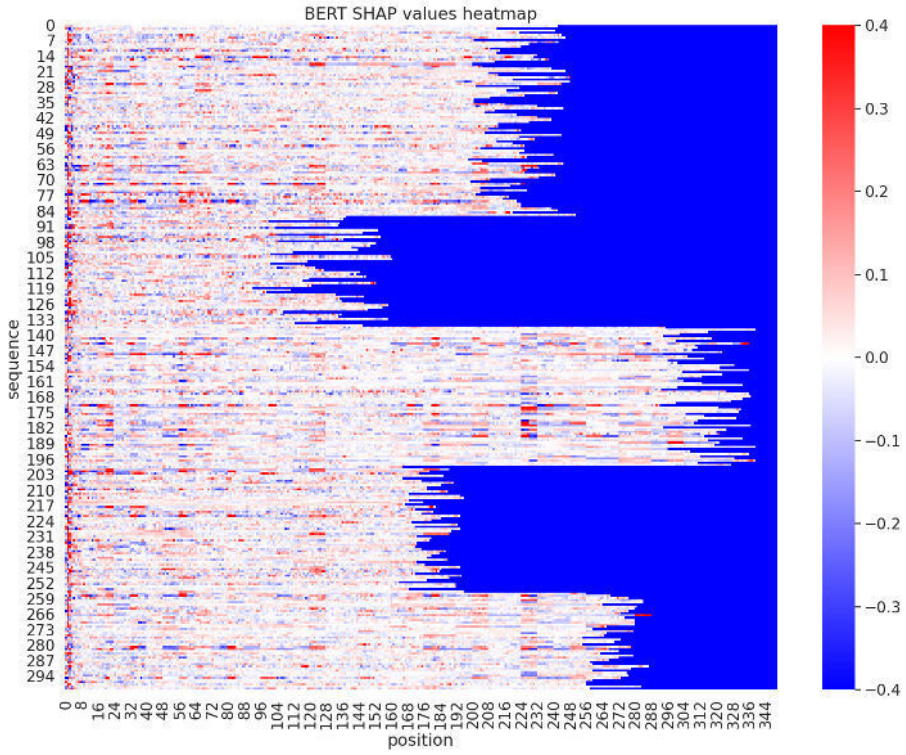

Figure 7.5: Sequences shorter than 350 bps were padded with high negative values to clearly distinguish them

The positional analysis using SHAP does not distinguish a clear pattern, as was the case when analyzing the BERT attention scores.

I tried to analyze specific kmer SHAP values to see whether any 6-mer stands out as being highly specific to LTRs.

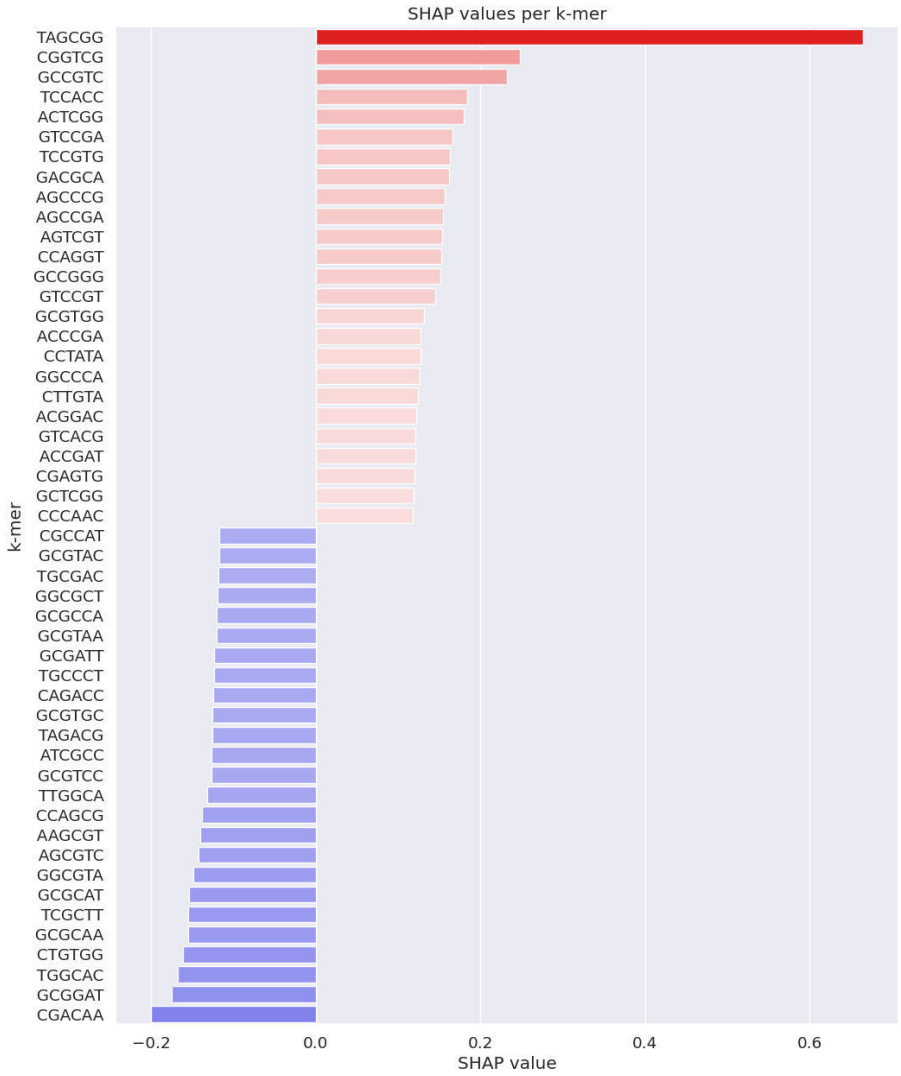

Figure 7.6: Top 50 k-mers, 25 with highest and 25 with lowest SHAP values

No typical promoter or LTR sequences such as CAAT-boxes or polyadenylation signals emerge here, however, the **CCTATA**, 6-mer could be of interest as it contains a sequence similar to the TATA-box (TATAWAW). Many of the top 6-mers form parts of known TF binding sites observed in a study of the architecture of promoters[69]: the **AGCGG** section of TAGCGG forms the initial part of TF motifs such

as ERF, a pathogenesis and stress-related promoter[70] and MYB3R1, a transcriptional repressor regulating organ growth[71]. The **GCCGT** sequence of GCCGTC can be mapped to zinc-finger and GCC-box like motifs.

Despite not being explicitly trained on TF binding site sequences, the trained BERT model seems to have implicitly fitted on 6-mers belonging to such motifs, further underlining their significance in LTR sequences.

8 Results

8.1 Testing model performance

In order to test the performance and versatility of the trained model, a special dataset was constructed, consisting of sequences from completely different plant species than the dataset on which the models have been trained. This dataset consisted of 5506 LTR sequences and 5506 non-LTR sequences, of which 25% were randomly generated sequences and 75% were genomic extracts mapped to non-LTR regions.

Species	LTR sequence count
Lemna Minor	1511
Arabidopsis Lyrata	2854
Carica Papaya	1141

Table 8.1: LTR sequences obtained from the different plant species, based on annotations from S. Zhou et al. [20] used for final testing

Once the database was created, the three model types (GB Classifier, NN, BERT) were used to draw predictions from the raw sequences. A python Class was created for each of the three model types, which, in the case of the latter two models, contains all three types of classifiers for different length sizes. Within the object, the different sequences are split into buckets (<350, 350-700, >700) to enable batch processing on GPUs. This provides users with an interface, omitting the need to split their dataset or preprocess it in any way, and speeds up the classification.

After drawing predictions, the models were evaluated using the accuracy, precision, and recall metrics. The dataset is balanced by design, so accuracy was a sufficient measure in this case. Precision and recall were examined to define the model's performance in real-life scenarios. One such scenario is the verification of annotations by TE recognition tools like the TE-greedy-nester[72]. In that case, the models should be able to detect falsely positive non-LTR sequences reliably.

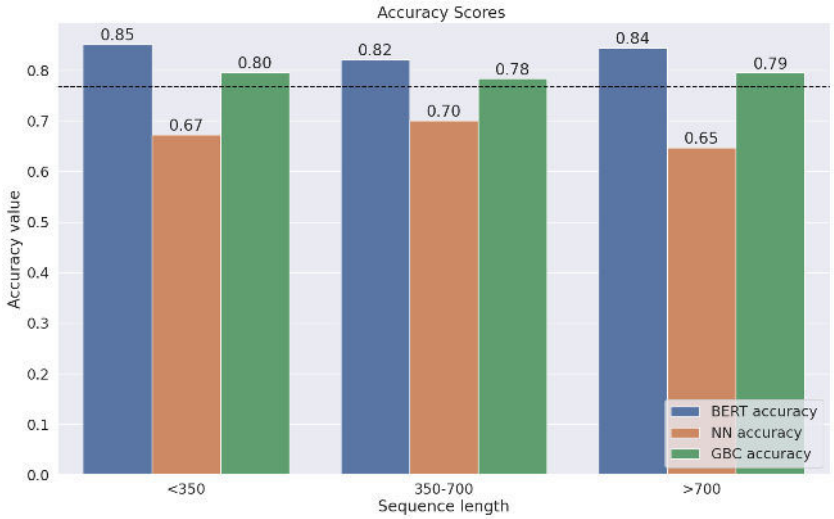

Figure 8.1: Accuracy of the three models on different length groups. The horizontal dotted line depicts mean accuracy across all three models

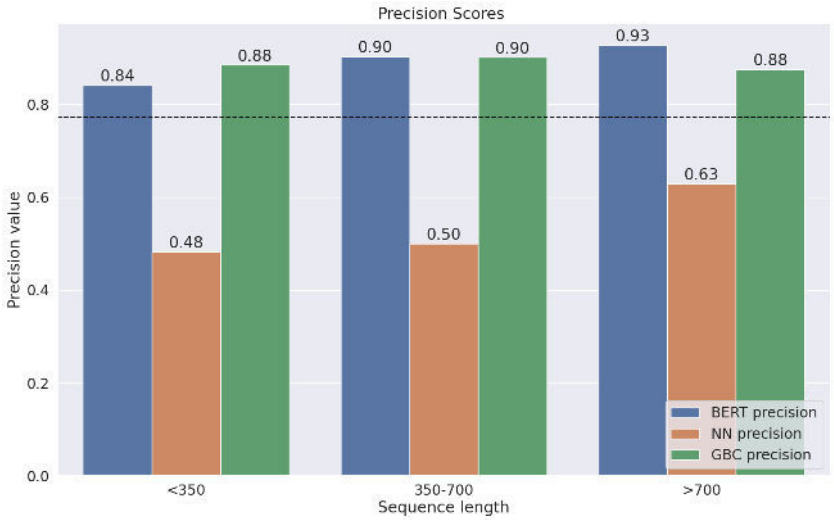

Figure 8.2: Precision of the three models on different length groups. The horizontal dotted line depicts mean precision across all three models

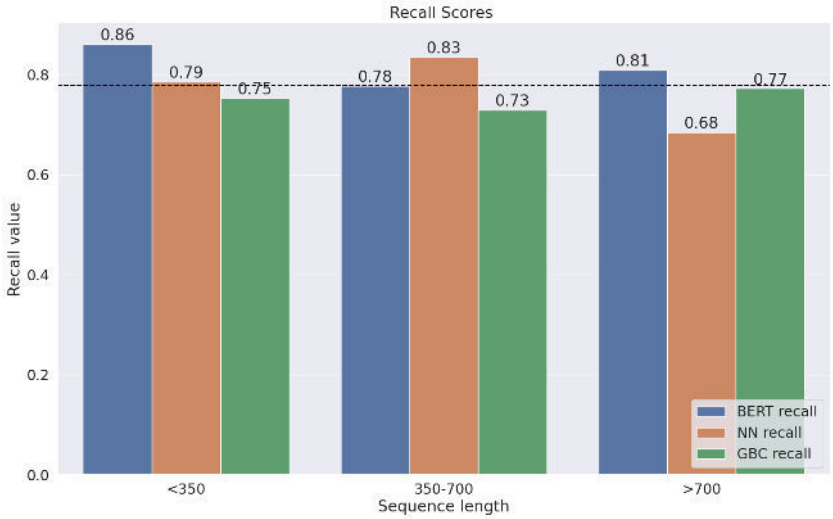

Figure 8.3: Recall of the three models on different length groups. The horizontal dotted line depicts mean recall across all three models

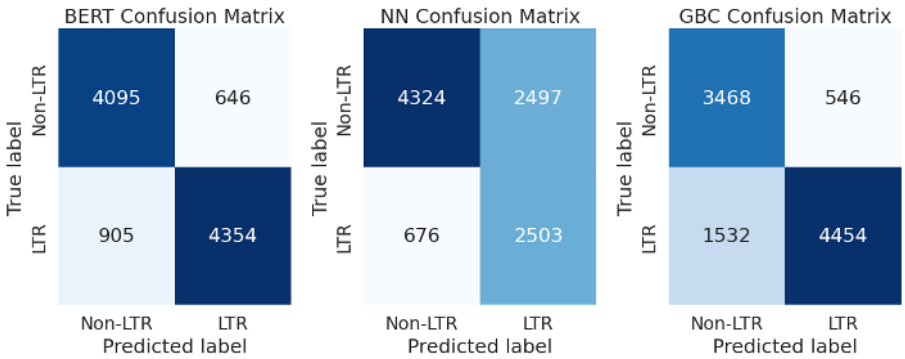

Figure 8.4: Confusion matrices of the three models

In general, the BERT classifier achieves the best results with an average accuracy of $\sim$ 84%; however, the RFC did better in terms of precision, achieving **0.94** precision score. The lowest performing is the neural network model, which trained and performed reasonably

well on a specific species dataset but struggled to generalize to previously unseen species' sequences. The NN model can detect non-LTR sequences even better than the other two classifiers as seen in 8.4, however, it misclassifies LTR sequences as non-LTRs in $\sim$ 50% of cases. In the binary classification scenario, this is equivalent to an untrained model.

8.1.1 Model performance on family and lineage subgroups

To verify if any models do not prioritize a certain family or lineage of TEs, I looked at the amount of correctly classified elements per each category by each of the three models. This test was done to see if LTRs belonging to a particular family or lineage significantly differ from others in a way that could impact classification using one of the three approaches.

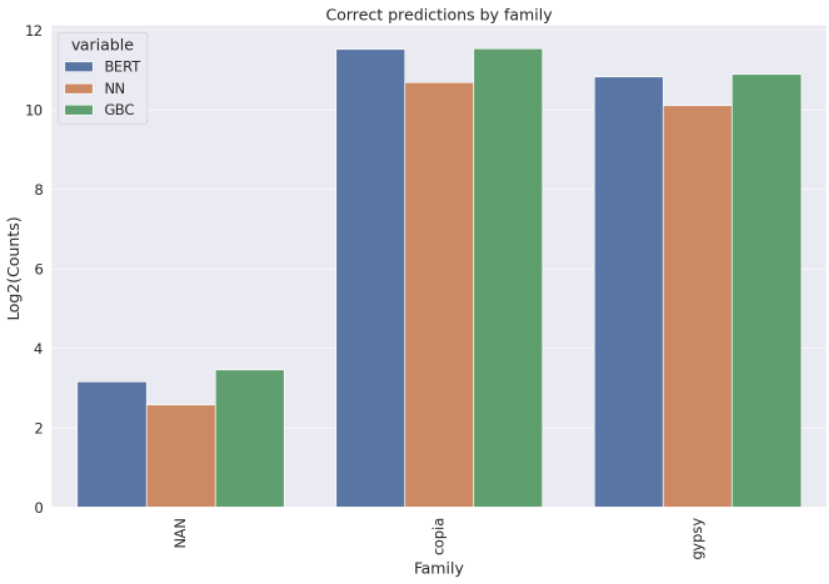

Figure 8.5: Log-normalized counts by LTR-TE family

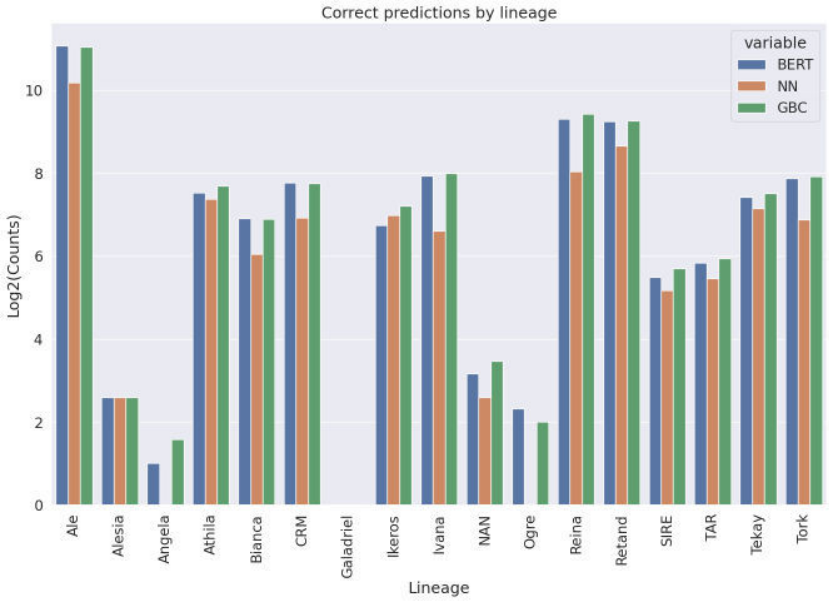

Figure 8.6: Log-normalized counts by LTR-TE lineage

Visually, lineage and family do not seem to impact model performance significantly. To statistically verify this, I conducted an ANOVA test, which returned a p-value of **0.713** for lineage and **0.971** for family. At a significance level of 0.05, the test fails to reject the null hypothesis that the means among the three groups are the same.

8.1.2 Tool comparison

In order to give an idea of the model's performance, I compared it to existing similar tools. As there is no tool for sole LTR classification, results were compared to existing promoter classification models, namely iProm-Zea[73], a CNN-based classifier specifically built for plant promoter recognition, iPromoter-2L[74] a Random Forest-based promoter recognition tool and iPTT(2L)-CNN[75], another CNN-based plant-promoter recognition tool.

Model	Acc(%)	Sn(%)	Sp(%)	MCC	Inp. Length
LTR_BERT	84.16	81.63	86.65	0.6839	< 5000
IPromoter-2L	81.7	79.2	84.2	0.637	81
iProm-Zea	96.06	97.95	87.65	0.8606	251
iPTT(2L)-CNN	94.70	87.81	87.81	0.8207	251

Table 8.2: Model comparison table. **Acc**: Accuracy, **Sn**: Sensitivity, **Sp**: Specificity, **MCC**: Matthew's correlation coefficient

The LTR_BERT model that I developed scores slightly better than IPromoter-2L and worse than iProm-Zea and iPTT(2L)-CNN, however, is applicable to an arbitrary input length sequence under 5000bps, whereas the other three models take fixed size inputs. This is a required property, as LTR sequences are more variable in length than promoters. Furthermore, both the iProm-Zea and iPTT(2L)-CNN models focus on a narrow range of species, namely *Zea Mays*, and in the case of iPTT(2L)-CNN, also *Arabidopsis Thaliana* and *Mus Musculus*, whereas LTR_BERT has been shown to achieve good results among various species.

9 Conclusion

The goal of this work was the development of a reliable training database, the analysis of LTR sequences and their function, and the classification of these elements based on their DNA sequence.

Based on frequent pattern mining of transcription factor motifs in LTR sequences, I concluded that common pairs, triplets, and quadruplets often share functionalities that can be linked to stress-induced processes and signaling pathways in plants.

Similar motifs emerged when conducting the SHAP model interpretation, further confirming this hypothesis.

I tested three classification approaches in order to select the best one to be used in practice. These included: a model trained on transcription factor binding site occurrences, a neural network model trained on one-hot encoded sequences, and a BERT transformer model.

Out of these classifiers, the BERT approach achieves the best scores overall.

Further analysis of the trained BERT model uncovered areas of significant attention values in the initial region and towards the ends of LTR, meaning these areas may be specific to these sequences. One sequence group proved to be different from the others in terms of high-attention locations, indicating that a smaller portion of LTRs may have a different distribution of important sequence motifs.

In comparison to similar tools the classifier developed in this thesis achieves slightly worse results overall, however, is more versatile than the other tools mentioned.

10 Deployment and Future work

The work is available at the LTR_BERT GitLab repository with instructions for its installation.

The separate trained BERT models are available from the HuggingFace website at:
xhorvat9/LTR_BERT_0_350_noTSD
xhorvat9/LTR_BERT_512_noTSD

The interface created currently supports discrete class predictions, except for the GBC pipeline, which comes with the *predict_proba* function by default.

I aim to extend the *BERT_predictor* class with logit returning functionality so that it may be used to evaluate the probability of sequences being LTRs in common LTR-TE detection tools. Tools such as the mentioned TE-Greedy-Nester program use inner scoring to rank detected TEs. This approach could utilize the LTR probability as a secondary weighing value to rerank the candidate elements.

Bibliography

1. THOMPSON, Peter J.; MACFARLAN, Todd S.; LORINCZ, Matthew C. Long Terminal Repeats: From Parasitic Elements to Building Blocks of the Transcriptional Regulatory Repertoire. *Molecular Cell*. 2016, vol. 62, no. 5, pp. 766–776. Available from DOI: 10.1016/j.molcel.2016.03.029.
2. HUGHES, Stephen H. Reverse Transcription of Retroviruses and LTR Retrotransposons. *Microbiology Spectrum*. 2015, vol. 3, no. 2. Available from DOI: 10.1128/microbiolspec.mdna3-0027-2014.
3. ALZOHAIRY, Ahmed; SABIR, J.S.M.; GYULAI, G.; YOUNIS, Rania; JANSEN, Robert; BAHIELDIN, Ahmed. Environmental Stress Activation of Plant LTR-Retrotransposons. *Functional Plant Biology*. 2014.
4. ROQUIS, David; ROBERTSON, Marta; YU, Liang; THIEME, Michael; JULKOWSKA, Magdalena; BUCHER, Etienne. Genomic impact of stress-induced transposable element mobility in Arabidopsis. *Nucleic Acids Research*. 2021, vol. 49, no. 18, pp. 10431–10447. Available from DOI: 10.1093/nar/gkab828.
5. MOLNAR, Christoph. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. Available also from: https://christophm.github.io/interpretable-ml-book.
6. LLORENS, Carlos; FUTAMI, Ricardo; COVELLI, Laura; DOMÍNGUEZ-ESCRIBÁ, Laura; VIU, Jose M.; TAMARIT, Daniel; AGUILAR-RODRÍGUEZ, Jose; VICENTE-RIPOLLES, Miguel; FUSTER, Gonzalo; BERNET, Guillermo P.; MAUMUS, Florian; MUNOZ-POMER, Alfonso; SEMPERE, Jose M.; LATORRE, Amparo; MOYA, Andres. The Gypsy Database (GyDB) of mobile genetic elements: release 2.0. *Nucleic Acids Research*. 2010, vol. 39, no. suppl$_1$, pp. D70–D74. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkq1061.
7. DU, Jianchang; GRANT, David; TIAN, Zhixi; NELSON, Rex T; ZHU, Liucun; SHOEMAKER, Randy C; MA, Jianxin. SoyTEdb: a comprehensive database of transposable elements in the soybean

genome. *BMC Genomics*. 2010, vol. 11, no. 1. Available from DOI: 10.1186/1471-2164-11-113.

8. SCHLAGENHAUF, Edith. *TREP, the transposable elements platform*. University of Zurich, [n.d.]. Available also from: https://trep-db.uzh.ch/.
9. OROZCO-ARIAS, Simon; JAIMES, Paula A.; CANDAMIL, Mariana S.; JIMÉNEZ-VARÓN, Cristian Felipe; TABARES-SOTO, Reinel; ISAZA, Gustavo; GUYOT, Romain. InpactorDB: A Classified Lineage-Level Plant LTR Retrotransposon Reference Library for Free-Alignment Methods Based on Machine Learning. *Genes*. 2021, vol. 12, no. 2, p. 190. Available from DOI: 10.3390/genes12020190.
10. XU, Zhenzhen; LIU, Jing; NI, Wanchao; PENG, Zhen; GUO, Yue; YE, Wuwei; HUANG, Fang; ZHANG, Xianggui; XU, Peng; GUO, Qi; SHEN, Xinlian; DU, Jianchang. GrTEdb: the first web-based database of transposable elements in cotton (Gossypium raimondii). *Database*. 2017, vol. 2017. Available from DOI: 10.1093/database/bax013.
11. XU, Zhao; WANG, Hao. LTR_FINDER: An efficient tool for the prediction of full-length LTR retrotransposons. *Nucleic Acids Research*. 2007, vol. 35, no. Web Server. Available from DOI: 10.1093/nar/gkm286.
12. ELLINGHAUS, David; KURTZ, Stefan; WILLHOEFT, Ute. LTRharvest, an efficient and flexible software for de novo detection of LTR retrotransposons. *BMC Bioinformatics*. 2008, vol. 9, no. 1. Available from DOI: 10.1186/1471-2105-9-18.
13. FU, Limin; NIU, Beifang; ZHU, Zhengwei; WU, Sitao; LI, Weizhong. CD-hit: Accelerated for clustering the next-generation sequencing data. *Bioinformatics*. 2012, vol. 28, no. 23, pp. 3150–3152. Available from DOI: 10.1093/bioinformatics/bts565.
14. HEALEY, James R. J. *Bioinfo-Tools: A Collection of Bioinformatics Tools* [https://github.com/jrjhealey/bioinfo-tools]. 2021. Accessed: 14-02-2023.

15. YE, Gongfu; ZHANG, Hangxiao; CHEN, Bihua; NIE, Sen; LIU, Hai; GAO, Wei; WANG, Huiyuan; GAO, Yubang; GU, Lianfeng. iDe novo/i genome assembly of the stress tolerant forest species iCasuarina equisetifolia/i provides insight into secondary growth. *The Plant Journal*. 2019, vol. 97, no. 4, pp. 779–794. Available from DOI: 10.1111/tpj.14159.

16. GUO, Shaogui; ZHANG, Jianguo; SUN, Honghe; SALSE, Jerome; LUCAS, William J; ZHANG, Haiying; ZHENG, Yi; MAO, Linyong; REN, Yi; WANG, Zhiwen; MIN, Jiumeng; GUO, Xiaosen; MURAT, Florent; HAM, Byung-Kook; ZHANG, Zhaoliang; GAO, Shan; HUANG, Mingyun; XU, Yimin; ZHONG, Silin; BOMBARELY, Aureliano; MUELLER, Lukas A; ZHAO, Hong; HE, Hongju; ZHANG, Yan; ZHANG, Zhonghua; HUANG, Sanwen; TAN, Tao; PANG, Erli; LIN, Kui; HU, Qun; KUANG, Hanhui; NI, Peixiang; WANG, Bo; LIU, Jingan; KOU, Qinghe; HOU, Wenju; ZOU, Xiaohua; JIANG, Jiao; GONG, Guoyi; KLEE, Kathrin; SCHOOF, Heiko; HUANG, Ying; HU, Xuesong; DONG, Shanshan; LIANG, Dequan; WANG, Juan; WU, Kui; XIA, Yang; ZHAO, Xiang; ZHENG, Zequn; XING, Miao; LIANG, Xinming; HUANG, Bangqing; LV, Tian; WANG, Junyi; YIN, Ye; YI, Hongping; LI, Ruiqiang; WU, Mingzhu; LEVI, Amnon; ZHANG, Xingping; GIOVANNONI, James J; WANG, Jun; LI, Yunfu; FEI, Zhangjun; XU, Yong. The draft genome of watermelon (Citrullus lanatus) and resequencing of 20 diverse accessions. *Nature Genetics*. 2012, vol. 45, no. 1, pp. 51–58. Available from DOI: 10.1038/ng.2470.

17. The International Barley Genome Sequencing Consortium. A physical, genetic and functional sequence assembly of the barley genome. *Nature*. 2012, vol. 491, no. 7426, pp. 711–716. Available from DOI: 10.1038/nature11543.

18. MARTINEZ-GARCIA, Pedro J.; CREPEAU, Marc W.; PUIU, Daniela; GONZALEZ-IBEAS, Daniel; WHALEN, Jeanne; STEVENS, Kristian A.; PAUL, Robin; BUTTERFIELD, Timothy S.; BRITTON, Monica T.; REAGAN, Russell L.; CHAKRABORTY, Sandeep; WALAWAGE, Sriema L.; VASQUEZ-GROSS, Hans A.; CARDENO, Charis; FAMULA, Randi A.; PRATT, Kevin; KURUGANTI, Sowmya; ARADHYA, Mallikarjuna K.; LESLIE, Charles A.; DANDEKAR, Abhaya M.; SALZBERG, Steven L.; WEGRZYN, Jill L.; LANG-

LEY, Charles H.; NEALE, David B. The walnut (iJuglans regia/i) genome sequence reveals diversity in genes coding for the biosynthesis of non-structural polyphenols. *The Plant Journal*. 2016, vol. 87, no. 5, pp. 507–532. Available from DOI: 10.1111/tpj.13207.

19. SCHNABLE, Patrick S. et al. The B73 Maize Genome: Complexity, Diversity, and Dynamics. *Science*. 2009, vol. 326, no. 5956, pp. 1112–1115. Available from DOI: 10.1126/science.1178534.

20. ZHOU, Shan-Shan; YAN, Xue-Mei; ZHANG, Kai-Fu; LIU, Hui; XU, Jie; NIE, Shuai; JIA, Kai-Hua; JIAO, Si-Qian; ZHAO, Wei; ZHAO, You-Jie; AL., et. A comprehensive annotation dataset of intact LTR retrotransposons of 300 plant genomes. *Scientific Data*. 2021, vol. 8, no. 1. Available from DOI: 10.1038/s41597-021-00968-x.

21. YOUENS-CLARK, Ken. *Mastering python for bioinformatics*. Sebastopol, CA: O'Reilly Media, 2021.

22. TOVAR, Benjamin. *Introduction to markov chains and modeling DNA sequences in R: R-bloggers*. 2012. Available also from: https://www.r-bloggers.com/2012/04/introduction-to-markov-chains-and-modeling-dna-sequences-in-r/. Accessed 26-03-2023.

23. LEE, Tong Ihn; YOUNG, Richard A. Transcriptional Regulation and Its Misregulation in Disease. *Cell*. 2013, vol. 152, no. 6, pp. 1237–1251. Available from DOI: 10.1016/j.cell.2013.02.014.

24. CASTRO-MONDRAGON, Jaime A; RIUDAVETS-PUIG, Rafael; RAULUSEVICIUTE, Ieva; BERHANU LEMMA, Roza; TURCHI, Laura; BLANC-MATHIEU, Romain; LUCAS, Jeremy; BODDIE, Paul; KHAN, Aziz; MANOSALVA PÉREZ, Nicolás; FORNES, Oriol; LEUNG, Tiffany Y; AGUIRRE, Alejandro; HAMMAL, Fayrouz; SCHMELTER, Daniel; BARANASIC, Damir; BALLESTER, Benoit; SANDELIN, Albin; LENHARD, Boris; VANDEPOELE, Klaas; WASSERMAN, Wyeth W; PARCY, François; MATHELIER, Anthony. JASPAR 2022: the 9th release of the open-access database of transcription factor binding profiles. *Nucleic Acids Research*. 2021, vol. 50, no. D1, pp. D165–D173. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkab1113.

25. COCK, Peter JA; ANTAO, Tiago; CHANG, Jeffrey T; CHAPMAN, Brad A; COX, Cymon J; DALKE, Andrew; FRIEDBERG, Iddo; HAMELRYCK, Thomas; KAUFF, Frank; WILCZYNSKI, Bartek, et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*. 2009, vol. 25, no. 11, pp. 1422–1423.

26. STORMO, Gary. Modeling the specificity of protein-DNA interactions. *Quantitative Biology*. 2013, vol. 1, pp. 115–130. Available from DOI: 10.1007/s40484-013-0012-4.

27. AGRAWAL, Rakesh; SRIKANT, Ramakrishnan. Fast Algorithms for Mining Association Rules. 1994, pp. 487–499. ISBN 1-55860-153-8. Available from DOI: 10.5555/645920.672836.

28. ZAKI, Mohammed J. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*. 2000, vol. 12, no. 3, pp. 372–390. Available from DOI: 10.1109/69.846291.

29. MOHAMMED, Wria; RAHIM, Payman. Mining RDF (Linked Data) using Éclat algorithm. 2019, vol. 3, pp. 1–25. Available from DOI: 10.25098/3.2.21.

30. GUPTA, Alind. ECLAT algorithm. 2012. Available also from: https://www.geeksforgeeks.org/ml-eclat-algorithm/. Accessed 28-04-2023.

31. SONG, Qingxin; HUANG, Tien-Yu; YU, Helen H.; ANDO, Atsumi; MAS, Paloma; HA, Misook; CHEN, Z. Jeffrey. Diurnal regulation of SDG2 and JMJ14 by circadian clock oscillators orchestrates histone modification rhythms in Arabidopsis. *Genome Biology*. 2019, vol. 20, no. 1. Available from DOI: 10.1186/s13059-019-1777-1.

32. PRASAD, Kalika; ZHANG, Xiuwen; TOBÓN, Emilio; AMBROSE, Barbara A. The Arabidopsis B-sister MADS-box protein, GORDITA, represses fruit growth and contributes to integument development. *The Plant Journal*. 2010, vol. 62, no. 2, pp. 203–214. Available from DOI: 10.1111/j.1365-313x.2010.04139.x.

33. YOON, Hye-Kyung; KIM, Sang-Gyu; KIM, Sun-Young; PARK, Chung-Mo. Regulation of leaf senescence by NTL9-mediated osmotic stress signaling in Arabidopsis. *Mol. Cells*. 2008, vol. 25, no. 3, pp. 438–445.

34. TANG, Weining; PERRY, Sharyn E. Binding Site Selection for the Plant MADS Domain Protein AGL15. *Journal of Biological Chemistry*. 2003, vol. 278, no. 30, pp. 28154–28159. Available from DOI: 10.1074/jbc.m212976200.

35. CUI, Dayong; ZHAO, Jingbo; JING, Yanjun; FAN, Mingzhu; LIU, Jing; WANG, Zhicai; XIN, Wei; HU, Yuxin. The Arabidopsis IDD14, IDD15, and IDD16 Cooperatively Regulate Lateral Organ Morphogenesis and Gravitropism by Promoting Auxin Biosynthesis and Transport. *PLoS Genetics*. 2013, vol. 9, no. 9, e1003759. Available from DOI: 10.1371/journal.pgen.1003759.

36. YU, Diqiu; CHEN, Chunhong; CHEN, Zhixiang. Evidence for an Important Role of WRKY DNA Binding Proteins in the Regulation of iNPR1/i Gene Expression. *The Plant Cell*. 2001, vol. 13, no. 7, pp. 1527–1540. Available from DOI: 10.1105/tpc.010115.

37. LI, Hongmei; SUN, Jiaqiang; XU, Yingxiu; JIANG, Hongling; WU, Xiaoyan; LI, Chuanyou. The bHLH-type transcription factor AtAIB positively regulates ABA response in Arabidopsis. *Plant Molecular Biology*. 2007, vol. 65, no. 5, pp. 655–665. Available from DOI: 10.1007/s11103-007-9230-3.

38. LIU, Yawen; LI, Xu; LI, Kunwu; LIU, Hongtao; LIN, Chentao. Multiple bHLH Proteins form Heterodimers to Mediate CRY2-Dependent Regulation of Flowering-Time in Arabidopsis. *PLoS Genetics*. 2013, vol. 9, no. 10, e1003861. Available from DOI: 10.1371/journal.pgen.1003861.

39. CONSORTIUM, The UniProt. UniProt: the Universal Protein Knowledgebase in 2023. *Nucleic Acids Research*. 2022, vol. 51, no. D1, pp. D523–D531. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkac1052.

40. RAUDVERE, Uku; KOLBERG, Liis; KUZMIN, Ivan; ARAK, Tambet; ADLER, Priit; PETERSON, Hedi; VILO, Jaak. g:Profiler: a web server for functional enrichment analysis and conversions

of gene lists (2019 update). *Nucleic Acids Research*. 2019, vol. 47, no. W1, W191–W198. Available from DOI: 10.1093/nar/gkz369.

41. SALTON, Gerard; BUCKLEY, Christopher. Term-weighting approaches in automatic text retrieval. *Information Processing Management*. 1988, vol. 24, no. 5, pp. 513–523. ISSN 0306-4573. Available from DOI: https://doi.org/10.1016/0306-4573(88)90021-0.

42. PEDREGOSA, Fabian; VAROQUAUX, Gaël; GRAMFORT, Alexandre; MICHEL, Vincent; THIRION, Bertrand; GRISEL, Olivier; BLONDEL, Mathieu; PRETTENHOFER, Peter; WEISS, Ron; DUBOURG, Vincent, et al. Scikit-learn: Machine learning in Python. *Journal of machine learning research*. 2011, vol. 12, no. Oct, pp. 2825–2830.

43. LUNDBERG, Scott M; LEE, Su-In. A Unified Approach to Interpreting Model Predictions. In: GUYON, I.; LUXBURG, U. V.; BENGIO, S.; WALLACH, H.; FERGUS, R.; VISHWANATHAN, S.; GARNETT, R. (eds.). *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 4765–4774. Available also from: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

44. NICOLAS, Michael; RODRÍGUEZ-BUEY, María Luisa; FRANCO-ZORRILLA, José Manuel; CUBAS, Pilar. A Recently Evolved Alternative Splice Site in the BRANCHED1a Gene Controls Potato Plant Architecture. *Current Biology*. 2015, vol. 25, no. 14, pp. 1799–1809. ISSN 0960-9822. Available from DOI: https://doi.org/10.1016/j.cub.2015.05.053.

45. FORNARA, Fabio; PANIGRAHI, Kishore C.S.; GISSOT, Lionel; SAUERBRUNN, Nicolas; RÜHL, Mark; JARILLO, José A.; COUPLAND, George. Arabidopsis DOF Transcription Factors Act Redundantly to Reduce CONSTANS Expression and Are Essential for a Photoperiodic Flowering Response. *Developmental Cell*. 2009, vol. 17, no. 1, pp. 75–86. Available from DOI: 10.1016/j.devcel.2009.06.015.

46. XIONG, Wenwei; WANG, Chunlei; ZHANG, Xiangbo; YANG, Qinghua; SHAO, Ruixin; LAI, Jinsheng; DU, Chunguang. Highly interwoven communities of a gene regulatory network unveil topologically important genes for maize seed development. *The*

Plant Journal. 2017, vol. 92, no. 6, pp. 1143–1156. Available from DOI: 10.1111/tpj.13750.

47. GAUDET, Pascale; LIVSTONE, Michael S.; LEWIS, Suzanna E.; THOMAS, Paul D. Phylogenetic-based propagation of functional annotations within the Gene Ontology consortium. *Briefings in Bioinformatics*. 2011, vol. 12, no. 5, pp. 449–462. Available from DOI: 10.1093/bib/bbr042.

48. HASSAN, Hala; SCHERES, Ben; BLILOU, Ikram. JACKDAW controls epidermal patterning in the iArabidopsis/i root meristem through a non-cell-autonomous mechanism. *Development*. 2010, vol. 137, no. 9, pp. 1523–1529. Available from DOI: 10.1242/dev.048777.

49. CHANG, Caren. Q&ampA: How do plants respond to ethylene and what is its importance? *BMC Biology*. 2016, vol. 14, no. 1. Available from DOI: 10.1186/s12915-016-0230-0.

50. AL-AJLAN, Amani; ALLALI, Achraf El. CNN-MGP: Convolutional Neural Networks for Metagenomics Gene Prediction. *Interdisciplinary Sciences: Computational Life Sciences*. 2018, vol. 11, no. 4, pp. 628–635. Available from DOI: 10.1007/s12539-018-0313-4.

51. GUNASEKARAN, Hemalatha; RAMALAKSHMI, K.; AROKIARAJ, A. Rex Macedo; KANMANI, S. Deepa; VENKATESAN, Chandran; DHAS, C. Suresh Gnana. Analysis of DNA Sequence Classification Using CNN and Hybrid Models. *Computational and Mathematical Methods in Medicine*. 2021, vol. 2021, pp. 1–12. Available from DOI: 10.1155/2021/1835056.

52. CÂMARA, Gabriel B. M.; COUTINHO, Maria G. F.; SILVA, Lucileide M. D. da; GADELHA, Walter V. do N.; TORQUATO, Matheus F.; BARBOSA, Raquel de M.; FERNANDES, Marcelo A. C. Convolutional Neural Network Applied to SARS-CoV-2 Sequence Classification. *Sensors*. 2022, vol. 22, no. 15, p. 5730. ISSN 1424-8220. Available from DOI: 10.3390/s22155730.

53. ZENG, Haoyang; EDWARDS, Matthew D.; LIU, Ge; GIFFORD, David K. Convolutional neural network architectures for predicting DNA–protein binding. *Bioinformatics*. 2016, vol. 32, no. 12, pp. i121–i127. Available from DOI: 10.1093/bioinformatics/btw255.

54. HAN, Guo-Sheng; LI, Qi; LI, Ying. Nucleosome positioning based on DNA sequence embedding and deep learning. *BMC Genomics*. 2022, vol. 23, no. S1. Available from DOI: 10.1186/s12864-022-08508-6.

55. KHANDAY, Owais; DADVANDIPOUR, Samad. Convolution Neural Networks and Impact of Filter Sizes on Image Classification. *Multidiszciplináris Tudományok*. 2020, vol. 10, pp. 55–60. Available from DOI: 10.35925/j.multi.2020.1.7.

56. *Long short-term memory networks (LSTM)- simply explained!* 2023. Available also from: https://databasecamp.de/en/ml/lstms.

57. OLAH, Christopher. *Understanding LSTM Networks* [https://colah.github.io/posts/2015-08-Understanding-LSTMs/]. 2015. Accessed: Apr 2023.

58. WANG, Kunfu; ZHANG, Pengyi; SU, Jian. A Text Classification Method Based on the Merge-LSTM-CNN Model. *Journal of Physics: Conference Series*. 2020, vol. 1646, no. 1, p. 012110. Available from DOI: 10.1088/1742-6596/1646/1/012110.

59. O'MALLEY, Tom; BURSZTEIN, Elie; LONG, James; CHOLLET, François; JIN, Haifeng; INVERNIZZI, Luca, et al. *KerasTuner* [https://github.com/keras-team/keras-tuner]. 2019.

60. CHOLLET, Francois et al. *Keras*. GitHub, 2015. Available also from: https://github.com/fchollet/keras.

61. BIEWALD, Lukas. *Experiment Tracking with Weights and Biases*. 2020. Available also from: https://www.wandb.com/. Software available from wandb.com.

62. DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*. 2018, vol. abs/1810.04805. Available from arXiv: 1810.04805.

63. LEE, Jinhyuk; YOON, Wonjin; KIM, Sungdong; KIM, Donghyeon; KIM, Sunkyu; SO, Chan Ho; KANG, Jaewoo. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *CoRR*. 2019, vol. abs/1901.08746. Available from arXiv: 1901.08746.

64. ROSSETTO, Allison; STEGMAN, Pierce. [N.d.]. Available also from: https://hitchhikers.yext.com/blog/bert-strengths-and-weaknesses-34eabc48587b.

65. WAHAB, Abdul; TAYARA, Hilal; XUAN, Zhenyu; CHONG, Kil To. DNA sequences performs as natural language processing by exploiting deep learning algorithm for the identification of N4-methylcytosine. *Scientific Reports*. 2021, vol. 11, no. 1. Available from DOI: 10.1038/s41598-020-80430-x.

66. DNA_bert_6 Pre-trained model. In: [https://huggingface.co/zhihan1996/DNA_bert_6]. [N.d.]. Accessed: 2023-04-15.

67. LUNDBERG, Scott. *Text Plots - SHAP Documentation* [https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/text.html?highlight=text%20plot]. 2021. Accessed on March 9, 2023.

68. WOLF, Thomas; DEBUT, Lysandre; SANH, Victor; CHAUMOND, Julien; DELANGUE, Clement; MOI, Anthony; CISTAC, Pierric; RAULT, Tim; LOUF, Rémi; FUNTOWICZ, Morgan; DAVISON, Joe; SHLEIFER, Sam; PLATEN, Patrick von; BEYSSAC, Clara. *Hugging Face's Transformers: State-of-the-art Natural Language Processing* [https://huggingface.co]. 2019. Online; accessed [April 16, 2023].

69. MOHANTY, Bijayalaxmi. Genomic architecture of promoters and transcriptional regulation of candidate genes in rice involved in tolerance to anaerobic germination. *Current Plant Biology*. 2022, vol. 29, p. 100236. ISSN 2214-6628. Available from DOI: https://doi.org/10.1016/j.cpb.2022.100236.

70. FUJIMOTO, Susan Y.; OHTA, Masaru; USUI, Akemi; SHINSHI, Hideaki; OHME-TAKAGI, Masaru. Arabidopsis Ethylene-Responsive Element Binding Factors Act as Transcriptional Activators or Repressors of GCC Box–Mediated Gene Expression. *The Plant Cell*. 2000, vol. 12, no. 3, pp. 393–404. Available from DOI: 10.1105/tpc.12.3.393.

71. KOBAYASHI, Kosuke; SUZUKI, Toshiya; IWATA, Eriko; NAKAMICHI, Norihito; SUZUKI, Takamasa; CHEN, Poyu; OHTANI, Misato; ISHIDA, Takashi; HOSOYA, Hanako; MÜLLER, Sabine; LEVICZKY, Tünde; PETTKÓ-SZANDTNER, Aladár; DARULA, Zsuz-

sanna; IWAMOTO, Akitoshi; NOMOTO, Mika; TADA, Yasuomi; HIGASHIYAMA, Tetsuya; DEMURA, Taku; DOONAN, John H; HAUSER, Marie-Theres; SUGIMOTO, Keiko; UMEDA, Masaaki; MAGYAR, Zoltán; BÖGRE, László; ITO, Masaki. Transcriptional repression by scpMYB/scp 3R proteins regulates plant organ growth. *The EMBO Journal*. 2015, vol. 34, no. 15, pp. 1992–2007. Available from DOI: 10.15252/embj.201490899.

72. LEXA, Matej; JEDLICKA, Pavel; VANAT, Ivan; CERVENANSKY, Michal; KEJNOVSKY, Eduard. TE-greedy-nester: structure-based detection of LTR retrotransposons and their nesting. *Bioinformatics*. 2020, vol. 36, no. 20, pp. 4991–4999. ISSN 1367-4803. Available from DOI: 10.1093/bioinformatics/btaa632.

73. KIM, Jeehong; SHUJAAT, Muhammad; TAYARA, Hilal. iPromZea: A two-layer model to identify plant promoters and their types using convolutional neural network. *Genomics*. 2022, vol. 114, no. 3, p. 110384. ISSN 0888-7543. Available from DOI: https://doi.org/10.1016/j.ygeno.2022.110384.

74. LIU, Bin; YANG, Fan; HUANG, De-Shuang; CHOU, Kuo-Chen. iPromoter-2L: a two-layer predictor for identifying promoters and their types by multi-window-based PseKNC. *Bioinformatics*. 2017, vol. 34, no. 1, pp. 33–40. ISSN 1367-4803. Available from DOI: 10.1093/bioinformatics/btx579.

75. SUN, Ang; XIAO, Xuan; XU, Zhaochun. iPTT(2L)-CNN: A Two-Layer Predictor for Identifying Promoters and Their Types in Plant Genomes by Convolutional Neural Network. *Computational and Mathematical Methods in Medicine*. 2021, vol. 2021, pp. 1–9. Available from DOI: 10.1155/2021/6636350.