NexTSS: An ML Pipeline for Transcription Start Site Annotation

Final Project Report
03713 - Bioinformatics Data Practicum
Ashita Jawali, Parker Simpson, Sanat Mishra, Sofia Lima

Introduction

Transcription start site (TSS) is the first nucleotide from which transcription is initiated. The process of transcription is essential for gene expression, begins at the transcription start site, where RNA polymerase binds to the DNA and initiates the synthesis of a complementary RNA molecule. Accurate identification of TSSs is crucial for understanding gene regulation, predicting gene structures, and designing therapeutic interventions for various genetic diseases. The intricate nature of transcription regulation lies in the fact that TSSs are not universally conserved across different organisms, and their locations can vary even within the same species. Furthermore, alternative transcription start sites can give rise to multiple transcript isoforms, adding another layer of complexity to the problem. Consequently, the reliable identification of TSSs is a critical step in annotating and understanding the functional regions of a genome.

Computational approaches to TSS detection have emerged as indispensable tools for the task, owing to the massive amounts of genomic data produced by high-throughput sequencing technologies. These methods are designed to identify TSSs by analyzing sequence patterns, motifs, and other features associated with the transcription initiation process. However, the problem remains challenging due to the vast size and complexity of genomic data, the presence of multiple TSSs for individual genes, and the interplay between different regulatory elements.

Machine learning techniques, particularly deep learning, have shown great promise in addressing such complex challenges. These methods can capture intricate patterns in the genomic data, learn relevant models from informative features, and provide a more accurate prediction of TSSs. Furthermore, the integration of various data types, such as chromatin accessibility and histone modification data, can improve the overall performance of these computational models.

Background

Our pipeline has been inspired from DeepTSS - a computational method for transcription start site (TSS) detection using a deep-learning model. The DeepTSS model is designed to capture complex patterns in genomic data and predict single-nucleotide sites of transcription initiation. An important aspect of DeepTSS is the integration of genomic feature data to improve the overall performance of the model.

Project Workflow

The workflow broadly consists of the three phases - i) Preprocessing, ii) Feature Extraction, and iii) Modeling as seen in Fig 1. In this section, some parts of the workflow are described. Each of the three parts are described in more detail in 'Implementation' in the Methods section.

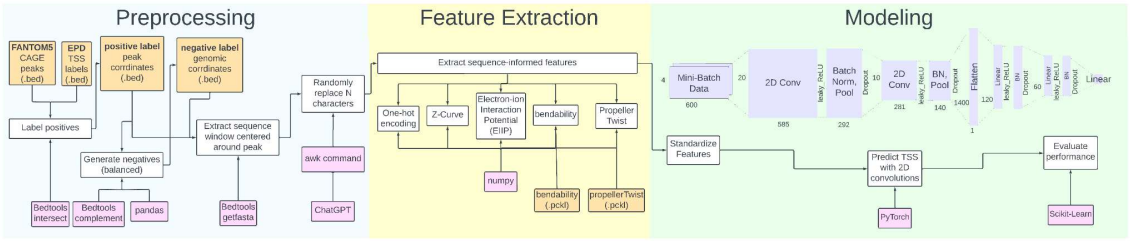

Fig 1. The complete workflow from preprocessing the data to making TSS predictions.White boxes are the computational objectives. Orange boxes are the datafiles needed for the pipeline. Pink boxes are the tools used to accomplish the corresponding task. Each of the three steps is described in more detail in the subsequent paragraphs.

Methods

Data

The raw data files used are listed below:

- TSS

The required TSS calls (from CAGE-seq data) can be downloaded from the following sources
i) EPD (Eukaryotic Promoter Database) (https://epd.epfl.ch//index.php)
ii) FANTOM (https://fantom.gsc.riken.jp/)
Data usage: Creates true labels for genuine TSS sites, through which we train the model

- Reference genome (Hg38)

The reference genome can be downloaded from the UCSC Genome Browser (https://hgdownload.soe.ucsc.edu/downloads.html)
Data usage: The reference genome enables us to select window sizes to look for TSS
While we have implemented the code for the Hg38 build, we can similarly do it for Hg19 if required using the UCSC liftOver tool

Implementation

PART I

Once the data files had been downloaded, we followed the processing steps shown on Fig. 2.

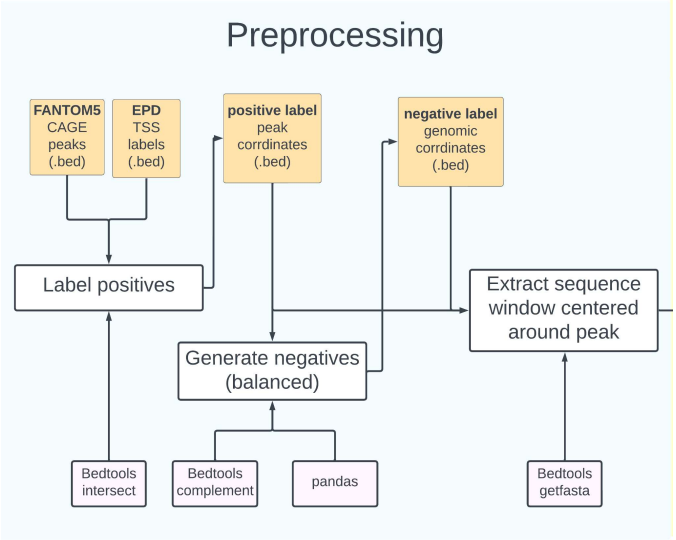

Fig. 2: The FANTOM and EPD databases are the canonical sources of TSS in *Homo sapiens.* The positive class data comes from these databases, while an equal number of negative class data was generated. A window around these positions was then used as input into the model.

Generating positively labeled data

First, we intersected the FANTOM and EPD datasets to find a list of positively labeled TSS regions. The EPD data consists of just a single nucleotide position that corresponds to the TSS, while the FANTOM data consists of genomic regions of varying base pairs in length. Since FANTOM consists of experimental CAGE-Seq data and EPD is a manually curated dataset, we decided that the best way forward was to select the elements common to the two databases for generating the positive labels to train the model.
Input: BED file from FANTOM and BED file from EPD
Output: BED file containing the intersected regions from the input files

Get genome sequences for positively labeled data

To extract the genomic region around the coordinates generated in the previous step, a **600 base** window centered at each single nucleotide entry from the previous file was created. Then, the genome sequence corresponding to these regions was extracted from a reference genome file.
Input: BED file containing a 600 base window (positive windows) and reference genome file
Output: FASTA file containing the genome sequences

Generate negatively labeled data

To get negatively labeled data, the first thing we did is generate the complement of the FANTOM BED file that had gone into making the positively labeled data. Any region that is not part of the FANTOM BED but is part of the reference genome, was included in the complementary region file.
Input: FANTOM BED file and reference genome file
Output: BED file containing the complement regions from the input files

Get genome sequences for negatively labeled data

From the BED file generated above, we randomly selected regions that span 600 bases. This was done counting the number of these regions per chromosome, such that there are an equal number

of positively and negatively labeled regions. This would ensure that the model is trained on a balanced dataset of equal numbers of positive and negatively labeled data.
Input: BED file containing complementary regions and BED file containing the positively labeled regions
Output: FASTA file containing negatively labeled data balanced with the positively labeled data such that the same number of negative samples are randomly selected from each chromosome

PART II

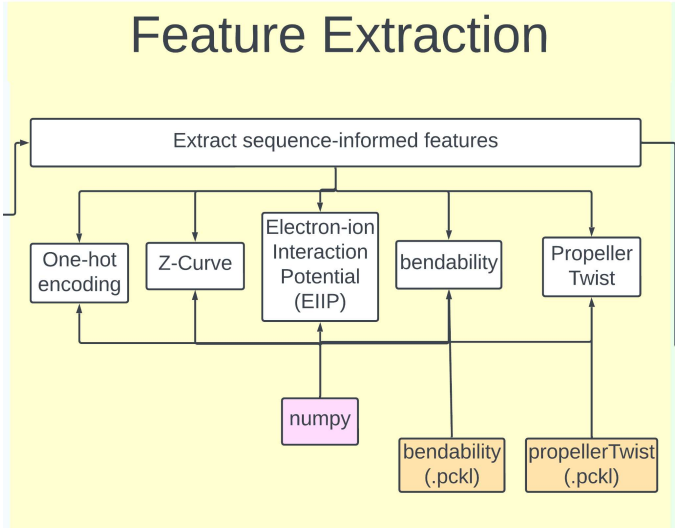

Fig. 3: The pipeline to extract sequence features. There are two feature categories: the one-hot encoded sequence, the genome signal features - Z-Curve, Electron-ion Interaction Potential, Bendability, and Propeller Twist.

One-hot encoding
It is a technique used in data preprocessing to convert categorical variables into numerical format without biasing the relative order of non-numerical data that can be used for machine learning algorithms.

Z-Curve
It is a bioinformatics tool which is used for the analysis of the DNA sequence. It is a mathematical method that transforms a DNA sequence into a three-dimensional curve, which can then be used to analyze various features of the sequence, such as base composition, periodicity, and symmetry.

Electron-ion Interaction Potential (EIIP)
Electron-ion Interaction Potential (EIIP) is a concept used in computational biology and bioinformatics to assign numerical values to the four nucleotide bases of DNA and the 20 amino acids of proteins. The EIIP value is a measure of the average energy required to ionize an atom of the base or amino acid, and it is calculated using quantum mechanical models.

Bendability
It is used to describe the flexibility or deformability of DNA molecules. Specifically, it refers to the propensity of a particular DNA sequence to bend or deform in response to external forces or binding interactions with other molecules.

Propeller Twist
It describes the twisting of adjacent base pairs in a DNA or RNA helix. Specifically, it refers to the angle of rotation between the planes of two adjacent base pairs in the helix.

PART III

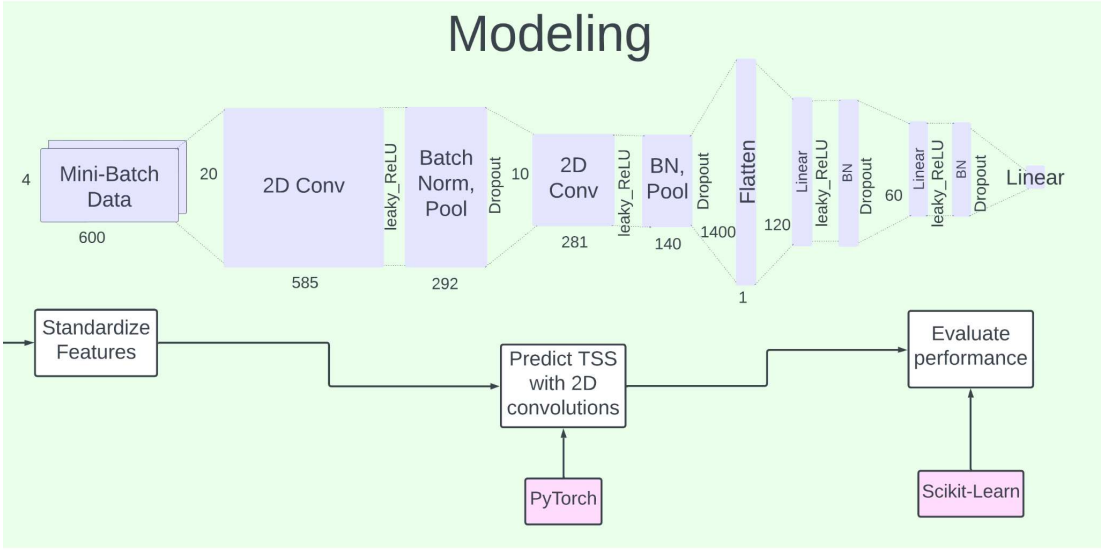

Fig. 4: The model architecture of the CNN. The various parts of it are shown in lilac.

The model developed in this project was inspired by the CNN-based architecture of DeepTSS, where each sample consists of a matrix that has size (number of features, window length). While DeepTSS trains four separate branches of 1D convolutional layers followed by a concatenation as shown in the model 28-09.hdf5 file on github, our model utilizes 2D convolution in the hopes of capturing relative information among the sequence-informed features. We use the same neural network operations: two convolutional layers with kernel sizes (number of features, 16) then (number of features, 12) respectively, each followed by leaky_ReLU activation, batch normalization, 2D max pooling with kernel size (2,2), and 30% dropout to reduce overfitting; then a flattening operation followed by 2 linear layers with out_features sizes 120 and 60 respectively, each followed by leaky_ReLU activate, batch normalization, and 30% dropout; and finally a linear layer with out_features size 1 and followed by sigmoid activation. This provides the probability of TSS for each sample. We use binary cross entropy as our loss to minimize, and we use the Adam optimizer with default learning rate 0.001.

From a technical standpoint, we make certain considerations to implement the training and testing of our model. When using pytorch 2D convolutional layers, the input must be 3 dimensions (the third dimension is typical of channels in imaging data) so we had to expand the dimension such that each sample has 1 channel. Given our large sample size of over 46000 for humans and 30000 for mice, we use mini-batch for data loading and training with batch size of 512 for humans and 256 for mice. We found that using GPU with cuda was much faster than CPU nodes for running our model. We chose accuracy, F1 score, precision and recall as our metrics for evaluating the performance of our model.

Results

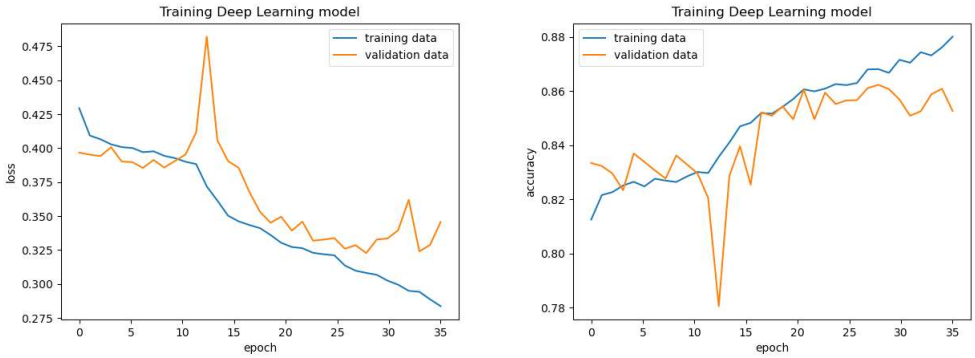

Fig. 5: Training and validation Binary Cross Entropy loss (left) and accuracy (right)

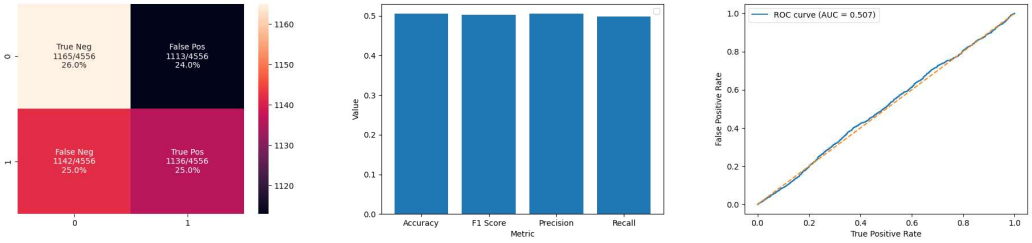

Fig. 6: Evaluation on human data. Confusion matrix (top left), Classification metrics (top right), ROC (bottom)

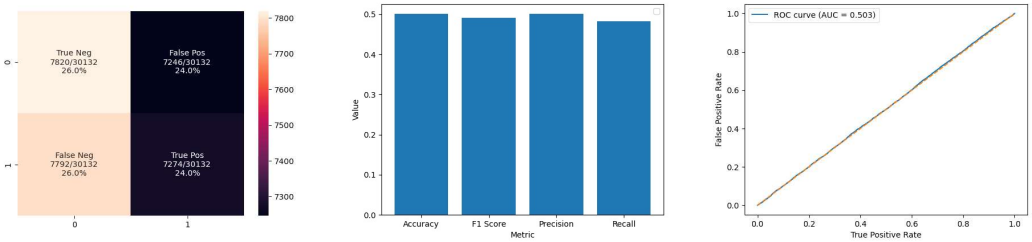

Fig. 7: Evaluation on mouse data. Confusion matrix (top left), Classification metrics (top right), ROC (bottom)

We used a 70/20/10 train/validation/test split when training the model on the human data. In figure 5 we can observe the training and validation loss and accuracy over 35 epochs of training. According to this graph the model seems to be learning well with a validation accuracy of ~85% at the end of training. However, when we evaluate the model on the human data and the mouse data we observe a vastly different outcome. As shown in the confusion matrices in figures 6 and 7, our model produces an almost even amount of true vs false predictions. Because all splits of our data are evenly class balanced, this indicates that our model is performing evenly with random guessing.

Discussion

Model Limitations

Our results show the overfitting of our deep learning model to the data trained with mini-batches. We see that this training scheme initially begins with around 80% accuracy on the training data. Our testing accuracy never reaches this level because the high accuracy even at the beginning of training is expected from training in batch where the first report of accuracy is the result of the model that has already trained several rounds. The continual increase in accuracy shows the further overfitting. Thus, our testing accuracy remains quite low due to overfitting to training data–a vulnerability especially with deep learning models.

While generating random sequences as negative samples could improve accuracy due to differences in GC content, the practical problem of improving SNR in CAGE-data should be addressed with samples that are hard to distinguish. To improve the accuracy, we could use the CAGE peaks that were not found to intersect EPD as the ground truth negative samples. We could also fine-tune the hyperparameters of the DL model. Importantly, it would be worth exploring other generalizable models such as SVM and RF.

Future directions

Our main objective in the pipeline was to minimize the noise in the CAGE-seq data and increase the signal-to-noise ratio. This would enable a more accurate framework for identifying TSS. To that end, we think that adding the CAGE-seq score data to the model will improve accuracy and give more insight into the CAGE-seq calls.

The model presented here utilizes genomic features from a species as training data and that from a different species to test model performance. While we did not use any particular conservation score as an input feature, we think that using features derived from raw sequences can work equally well. The reason is that, at the genomic level, the window-based approach is similar to looking at a smaller subsequence (akin to a large k-mer).The different GSP (Genome Signal Processing) scores that we're including in the model capture information that would have been conveyed via sequence conservation scores. However, we could add a conservation score metric just as is done in the original DeepTSS model.

The DeepTSS model that we followed used percentage overlap of predicted TSS sites with active and inactive chromatin states as a benchmarking method. Additionally, DeepTSS used transcription factors and H3K4Me3 occupancy sites as a method to benchmark their predictions. We could include the former method to our model to see how NexTSS compares with DeepTSS.

References

Grigoriadis, Dimitris, et al. "DeepTSS: multi-branch convolutional neural network for transcription start site identification from CAGE data." *BMC bioinformatics* 23.2 (2022): 1-17.

Mendizabal-Ruiz, Gerardo, et al. "Genomic signal processing for DNA sequence clustering." *PeerJ* 6 (2018): e4264.

Borrayo, Ernesto, et al. "Genomic signal processing methods for computation of alignment-free distances from DNA sequences." *PloS one* 9.11 (2014): e110954.

Kwan, Hon Keung, and Swarna Bai Arniker. "Numerical representation of DNA sequences." *2009 IEEE International Conference on Electro/Information Technology*. IEEE, 2009.

Randhawa, Gurjit S., Kathleen A. Hill, and Lila Kari. "ML-DSP: Machine Learning with Digital Signal Processing for ultrafast, accurate, and scalable genome classification at all taxonomic levels." *BMC genomics* 20 (2019): 1-21.

Bonidia, Robson P., et al. "MathFeature: feature extraction package for DNA, RNA and protein sequences based on mathematical descriptors." *Briefings in bioinformatics* 23.1 (2022): bbab434.

Berger, John A., et al. "Visualization and analysis of DNA sequences using DNA walks." *Journal of the Franklin Institute* 341.1-2 (2004): 37-53.

Bonidia, Robson Parmezan, et al. "Feature Extraction Approaches for Biological Sequences: A Comparative Study of Mathematical Models." *bioRxiv* (2020): 2020-06.

Hoang, Tung, Changchuan Yin, and Stephen S-T. Yau. "Numerical encoding of DNA sequences by chaos game representation with application in similarity comparison." *Genomics* 108.3-4 (2016): 134-142.