Model Serving for Transcription Factor Binding Site Prediction

Alyssa Morrow
akmorrow@berkeley.edu

Devin Petersohn
devin.petersohn@berkeley.edu

Anthony Joseph
adj@berkeley.edu

Nir Yosef
nir.yosef@berkeley.edu

ABSTRACT

Current pipelines for training machine learning models on genomic and epigenetic datasets lack resources to scale to current dataset sizes. Furthermore, pipelines for testing and evaluating these machine learning models are segmented, as they require users to separately test pipelines and evaluate their methods using disjunct software. In this paper we survey current workflows for training, testing and evaluating genomic machine learning models and provide an alternative solution to connecting the dots between these processes. More specifically, we present a complete fast and interactive pipeline that allows users to save prebuilt models and evaluate these models in a visual environment with low latency.

Keywords

Genomics; Model Serving; Transcription Factors

1. INTRODUCTION

Regulatory genomics is a field of research that works to connect the function of DNA sequence, DNA interacting proteins and cell environment with gene expression. Understanding and predicting regulatory function is important, as it allows clinicians to predict the effect of an individual's genetic variants on their health and well being. Currently, there exist many different computational methods for prediction of regulatory function, including convolutional neural networks and SVM classification problems (Alipanahi et. al. 2014, Kelley et. al. 2016). Datasets used to predict regulatory function are massive, reaching 15 TB in the ENCODE consortium (cite). However, these datasets are noisy, and thus analysis of regulatory predictions often require a human in the loop to visually interpret predictions against ground truth datasets produced in datasets such as ENCODE.

Current pipelines for training and analyzing regulatory genomic models include a segmented group of steps that are exclusive and require preprocessing between each stage. For example, the state-of-the-art pipeline involves executing a locally run algorithm, such as a kernel method or neural network, built for a single machine. Users who are interested in predicting on a particular sequence face multiple challenges. Because the state-of-the-art methods are not built to scale past a single machine, users must first sub-select a number of regions of the genome to train on specific to their hardware capabilities. After sub-selecting the data and training on that data, the user would then load these regions into a single node visualization tool, such as IGV, and repeat the preprocessing step to validate their model. A demonstration of the workflow pipeline required using currently available tools can be shown in Figure 1.

Currently, there is no infrastructure available for users to upload and predict on their sequences while receiving real time model serving. In practice, this infrastructure is not available, due in part to the difficulty in serving models at a large scale. Genomic data typically ranges on the order of 100's of MB to 10's of GB of data. The models are typically built with 10's of GB to 100's of GB of data, creating a unique challenge when trying to achieve interactive model serving speeds.

In the paper, we describe an end-to-end pipeline that efficiently computes and predicts on user provided sequences while maintaining seamless integration between processing steps. In our previous work, we use a scalable and parallel algorithm to efficiently compute protein binding sites without the use of specialized hardware (Morrow et. al. 2016). Here, we utilize this algorithm to predict from regulatory datasets, and provide a full analysis pipeline to visualize results. This novel pipeline allows users to interactively predict on regions of interest in a query genome. We compare our pipeline to the current state-of-the-art approaches on a single node, and then analyze pipeline latency at scale.

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

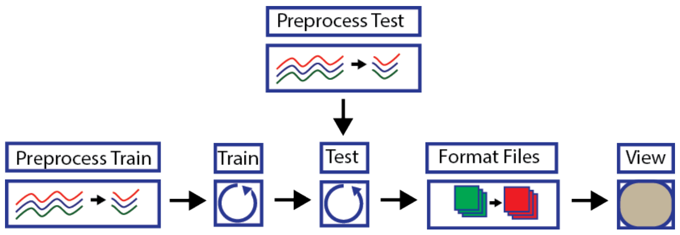


Figure 1: Stawman pipeline for training and evaluating model.

Table 1: Featurization Techniques

Featurization Technique	Datasets	Description	Usage
Read Wavelets (Raj et. al. 2015)	DNA-seq, RNA-seq	Iteratively quantifies read pileup	Identification of genomic footprints
Kernel Approximation (Morrow et. al. 2016)	Sequence, DNA-seq, RNA-seq	Approximates kernel matrix through random features	Generic, Widely applicable
K-mer counts	Sequence	Counts the number of times each motif appears in a sequence of length k	Identification of motifs, quantification of sequence similarity

2. BACKGROUND

Existing infrastructure for model prediction and serving is not built for scalable computing environments. In this section, we describe the current state-of-the-art pipeline components.

2.1 Algorithms

Current algorithms for predicting on regulatory genomic datasets include neural networks and SVM classification. These methods train on small portions of the genome. For example, a neural net implementation DeepBind sub-selects 10,000 sequences to predict protein binding sites. Similarly, Kernel SVM uses 10,000 sequence to learn a classifier. Both tools are intended to run on a single machine.

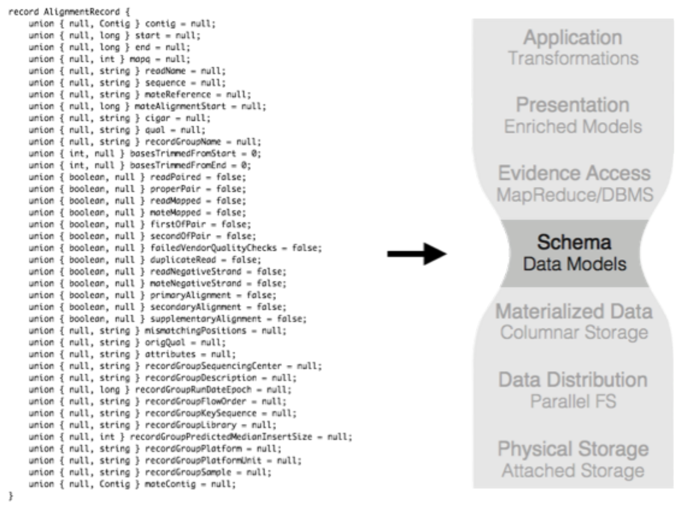
2.2 Evaluation Tools

Main interactive tools for evaluating algorithms discussed above include visualization tools, such as University of Santa Cruz Genome Browser and Integrated Genomics Viewer (IGV). IGV is a desktop application that allows visualization of a wide variety of genomic datasets. IGV is used by clinicians and researchers to verify anomalies in their data and verify predictions to original raw genomic datasets. IGV does not support integrated visualization of predictive models, so users attempting to perform visual analysis of models using heterogeneous datasets must predict outside of the visualization tool.

3. ARCHITECTURE

In this section, we outline our design for an end-to-end machine learning pipeline on epigenetic datasets. Our implementation builds upon previously built distributed genomics tools such as ADAM, Mango and Endive, which are described below. Finally, we will demonstrate how these tools can be connected together, with our new machine learning tool, to create an end-to-end pipeline for machine learning tasks.

3.1 ADAM



```
record AlignmentRecord {
    union { null, Contig } contig = null;
    union { null, long } start = null;
    union { null, long } end = null;
    union { null, int } mapq = null;
    union { null, string } readName = null;
    union { null, string } sequence = null;
    union { null, string } mateReference = null;
    union { null, long } mateAlignmentStart = null;
    union { null, string } cigar = null;
    union { null, string } qual = null;
    union { null, string } recordGroupName = null;
    union { int, null } basesTrimmedFromStart = 0;
    union { int, null } basesTrimmedFromEnd = 0;
    union { boolean, null } readPaired = false;
    union { boolean, null } properPair = false;
    union { boolean, null } readMapped = false;
    union { boolean, null } mateMapped = false;
    union { boolean, null } firstOfPair = false;
    union { boolean, null } secondOfPair = false;
    union { boolean, null } failedVendorQualityChecks = false;
    union { boolean, null } duplicateRead = false;
    union { boolean, null } readNegativeStrand = false;
    union { boolean, null } mateNegativeStrand = false;
    union { boolean, null } primaryAlignment = false;
    union { boolean, null } secondaryAlignment = false;
    union { boolean, null } supplementaryAlignment = false;
    union { null, string } mismatchingPositions = null;
    union { null, string } origQual = null;
    union { null, string } attributes = null;
    union { null, string } recordGroupSequencingCenter = null;
    union { null, string } recordGroupDescription = null;
    union { null, long } recordGroupRunDateEpoch = null;
    union { null, string } recordGroupFlowOrder = null;
    union { null, string } recordGroupKeySequence = null;
    union { null, string } recordGroupLibrary = null;
    union { null, int } recordGroupPredictedMedianInsertSize = null;
    union { null, string } recordGroupPlatform = null;
    union { null, string } recordGroupPlatformUnit = null;
    union { null, string } recordGroupSample = null;
    union { null, Contig } mateContig = null;
}
```

Figure 2: Example of AlignmentRecord schema used in Narrow waist design. Schema layer can be swapped in or Out for genomic workloads and substituted in for legacy file formats.

ADAM is a framework for processing genomic data, and is built to run in the Spark ecosystem. ADAM uses distributed abstractions in Spark to batch process raw genomic datasets in parallel. More importantly, ADAM provides standardized schemas intended for various types of genomic datasets, presented in a narrow waist design that can be easily accessible by external applications. A figure of schema design is shown in Figure X. Important schemas that are used for epigenetic datasets include Alignment Record, Features, and Variant schemas. These schemas are demonstrated in Figure 3.1.

3.2 Mango

Mango is a distributed genome visualization browser that selectively materializes and organizes genomic data to provide fast in-memory queries. Mango materializes data from persistent storage as the user requests different regions of the genome. This data is efficiently partitioned and organized in memory using a specialized RDD structure built upon interval trees. This interval based organizational structure supports ad hoc queries, filters, and joins across multiple files, enabling exploratory interaction with genomic data. Mango is built on top of Spark and ADAM. A demonstration of Mango architecture can be seen in Figure X. Mango is implemented in three layers, namely, the cluster, server and client. In the cluster layer, Mango is built on ADAM and Spark, allowing users to perform and visualize fine grained queries on large genomic files. The server layer reduces and samples queries so that users can visualize their queries in an efficient and interpretable manner.

Our machine learning tool, called Endive, provides core abstract stages for building end-to-end machine learning pipelines to train and evaluate on genomic datasets. These stages include data preprocessing, featurization, and various solvers apt for genomic workloads. These stages in a machine learning pipeline can be mixed and matched to create novel pipelines tailored to a genomic prediction task of interest. Preprocessing in Endive involves taking raw genomic data and parsing

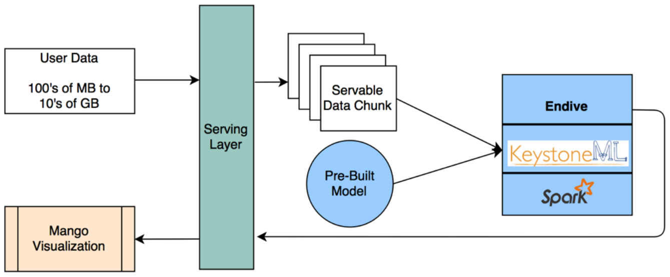


Figure 3: System Design overview.

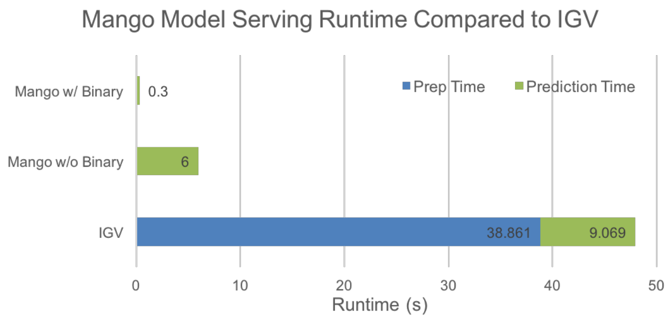


Figure 5: Runtime comparison with IGV and DeepBind

it into a vector of numeric values which can be directly read by a machine learning model. These preprocessing steps include formatting datasets as well as joining together heterogeneous datasets into one record based on coordinate location in the genome. Featurization steps in Endive involve taking preprocessed datasets created in the preprocessing stage and transforming these records to explicit features that can be trained and tested on. Table 1 contains examples of these featurization steps. These computed features can then be fed into a choice of solvers developed in Keystone ML to train these machine learning models (Sparks, 2015).

3.3 Machine Learning Pipeline for Epigenetic Workloads

For the entirety of this paper, we will focus on using kernel approximations described in Table 1 to predict protein binding sites for an example pipeline.

3.4 Interactive Prediction of Binding Sites

Initially, direct prediction of binding sites was not in interactive time (6s). To reduce latency, we took advantage that regulatory regions are fairly sparse, occurring in less than 0.01% of the entire genome. To take advantage of this sparsity, we iteratively analyzed the prediction of large potential regulatory regions with a pre-fetching technique. If this prediction scored higher than a given threshold, we cut the sequence into two equal parts iteratively pre-fetched the sequence cut in half, similar to a binary search implementation. This binary pre-fetching implementation allowed us to significantly reduce average prediction time of a servable chunk. A schematic of this approach is seen in Figure 3.4.

4. EXPERIMENTS

For preliminary results, we train a model predicting transcription factor binding sites using the kernel approximation featurization method described in Table 1 from ENCODE datasets using DNA sequence.

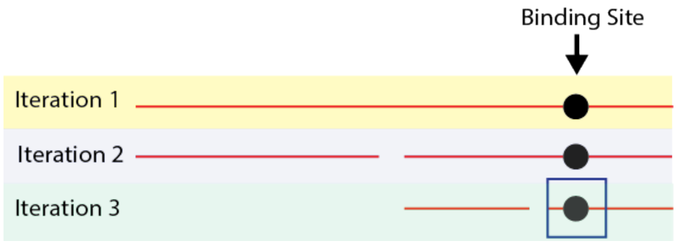


Figure 4: Binary Pre-fetching

4.1 Fast Prediction with Mango

For a straw man comparison, we took the state of the art pipeline used to train and analyze protein binding sites on a local computer. We trained a model on 15,000 DNA sequences to predict the location of binding sites of the EGR1 transcription factor, then visualizing the results in a genome browser, IGV. The state of the art in transcription factor binding is DeepBind, and uses a neural net. Previously, we demonstrated the improved accuracy and training time of the Convolutional Kitchen Sink over DeepBind's neural net implementation (Morrow et. al. 2016). The results of the runtime experiment for serving the model are displayed in Figure 4.1. A significant difference between Model Serving with Mango and using IGV is that with Mango, we are able to use Servable Chunks, which are defined in the user space. With IGV, there is no way to perform this prediction without completing the entire prediction over the whole search space.

Initially, our results were on the order of 6 seconds, which is sub-optimal for an application with a user interface. The results in Figure # show a difference between our implementation with and without the binary pre-fetching approach. Binary pre-fetching allows us to rule out a significant part of the Servable Chunk.

4.2 Whole Genome Model Serving

This model was solved using Keystone ML's Weighted Least Squares estimator, which accounts for large class imbalance of positive binding sites across the genome. This model was trained on 200,000,000 regions of the genome and generated a 4 MB model to be tested on new datasets. This model was loaded into the Mango genome browser and was used to interactively predict on DNA reference sequences as the user scrolled to new regions of the browser.

Table 2 shows the results of this experiment. Even on a significantly larger model, we averaged prediction times in the 400ms range. This is close to the level required for interactive applications, particularly applications with a user interface.

5. CONCLUSION

Table 2: Predicting on Entire Genome

Mean	Min	Max
402.31ms	3.09ms	3.98s

There is a need in the scientific community for a tool that accepts user data uploaded from a browser. Genomic data presents new challenges to model serving. Currently, there is no infrastructure in place to allow users to upload and interactively predict on their data. Because of the scale, interactivity is a difficult challenge. In this paper, we describe a novel approach to model serving for extremely large datasets. The process involves using Servable Chunks defined in user space to cut down on network and model prediction overheads. Additionally, we describe a binary prefetching approach to achieve close-to-interactive prediction times. Further optimization can be achieved in many parts of the pipeline. We are first interested in taking further advantage of sparsity of the positive predictions. We can do this in a couple of obvious ways. First, we can build a Hidden Markov Model (HMM) on the sequences and uploaded data, allowing us further optimization on the basis of where they will likely look for results next. With this information, we can further pre-fetch results to get them ready for the user's change in prediction window. Second, we can take a sampling method that does not examine the entire sequence. For this sampling method, we do not need to sample from a positive prediction to know that there is a positive prediction nearby. By using a sampling method, we can cut down on the size of the data being predicted and significantly reduce the prediction time. A combination of these two techniques with the Binary Pre-fetching is likely to improve the runtime of predictions to interactive levels.

There are several other future components we would like to add to this model serving pipeline. Another important aspect we would like to incorporate into our approach is a dynamic update component. When users upload their sequences, we want to improve our model with their data. Additionally, we would like to share the computation with users. If we can share some of the computation with users, we can significantly reduce the amount of network traffic and computation load on our server. There are several challenges with this model. First, users will have access to the prediction model, which can be problematic for security. In the case that we use private data for predicting these sequences, we do not want users to be able learn the original training data composition based on our model. Before we can split computation, we need a way to obscure the training data so that our private training data is not compromised.

In order to make this a usable application, we will need to develop the model serving application into a full pipeline. Specifically, we will need to include preprocessing for the data being uploaded so that it is always in the correct format. The field of genomics suffers from an incredibly wide range of formats and file types. This presents challenges when building tools; each format has nuances that require unique methods to handle them. Additionally, users should be able to download the full prediction over their entire sequence once it's complete.

6. REFERENCES

6.1 References

`\thebibliography`.