Applying Machine Learning to the Genome

Daniel Kang
Genome

- Base: character
- k-mer: string (length k)
- Genome: ~ 2.8 billion bases
DNAse-seq

Digest nuclei with DNase-I
(concentration/exposure specific)

Collect DNA, size separate (100-300bp)

DNase-seq read count:

Chromatin state:

Sequence (60-100M reads)
We can use DNAse-seq to predict functional genomic areas.
Prior work

- Binning + smoothing
Prior work: Limitations

- Are ad-hoc
- Require hand-tuned parameters
- Require resolution/noise tradeoff
- Low statistical power
- Focuses on specific parts of the genome
Model overview

- “cis-regulatory k-mer model”
- Every k-mer has an independent effect everywhere it appears in the genome
- Effects add in log space (exponential effect in read space)
- Poisson process
Model overview

Each Kmer alone

Combined effects

A). Basewise NOT using Kmer 3

B). Basewise AND using kmers 1 and 2

C). Basewise OR using the intersection kmer

Example Kmer

AACCCATCGTAGTCCCTTAGACT

Intersect Kmer (intersection of kmers 1 and 2)
Model benefits

● Parameter free
● Genome-wide
● Testable prediction
Model: Poisson process

- Log-Poisson rate: \( \lambda_i = \sum_{j=-W}^{W} u^k (g(i,k), j) - x_0 \)
- Log-likelihood: \( LH_i = c_i \lambda_i - \exp(\lambda_i) \)
- Objective function: \( F = -\sum_{i=1}^{N} LH_i + \eta (\sum_{k=1}^{8} \sum_{i=1}^{4^k} \sum_{j=1}^{2^W+1} |v^k_i[j]|) \)
Inference method: Gradient descent

\[
v^k_{g(i,k),j} = \begin{cases} 
0 & \text{if } |\hat{c}_{g(i,k),j} - c_{g(i,k),j}| < \eta \\
\ln \left( \frac{\hat{c}_{g(i,k),j} - \eta}{c_{g(i,k),j}} \right) & \text{if } \hat{c}_{g(i,k),j} > c_{g(i,k),j} \\
\ln \left( \frac{\hat{c}_{g(i,k),j} + \eta}{c_{g(i,k),j}} \right) & \text{else}
\end{cases}
\]
Serial implementation: Gradient descent

- Initialize parameter matrix $v$ to $0$
- Repeat until convergence:
  - Evaluate the gradient $dv$ at $v$
  - Update the parameter matrix via linear approximation: $v' = v + \varepsilon \ dv$
C++ threading

- pthreads
  - POSIX threads
  - Thread creation/management API
- OpenMP
  - Open Multi-Processing
  - API for shared memory multiprocessor programming
- MPI
  - Message Passing Interface
  - No shared memory model
MPI gotchas

- MPICH2 is faster than Open MPI
- MPI does not have a shared memory model
  - Locality aware bcast ~25% faster
  - Locality aware reduce ~5% faster
- Network communication is often the bottleneck
Parallel implementation: MPI

- Initialize nodes
- Initialize parameter matrix \( v \) to 0
- Repeat until convergence:
  - Send the current parameter vector to slaves
  - Each slave computes the gradient on a subset of the genome
  - The slaves send the gradient back to master, which then computes the full gradient \( dv \)
  - Master updates the parameter vector using the linear approximation \( v' = v + \varepsilon dv \)
Results: Synthetic Data

AGTCT

CAGAC
Results: Timings

NumPY implementation took ~300 minutes per iteration
C++ serial implementation took ~30 minutes per iteration

$R^2 = 0.85$
Future work

- Reduce communication time
- Further optimization
- Add features to the model
Acknowledgements

- David Gifford
- Tatsu Hashimoto
- Richard Sherwood