Sort-Seq Tools: sequence-function relationship modeling for massively parallel assays. Supplemental Information

William T. Ireland, Justin B. Kinney

May 20, 2016

1 Example pipeline

We now illustrate the capabilities of Sort-Seq Tools by stepping through an example pipeline. The reader is encourage to follow along by entering the commands shown. More information on each individual command can be found on the Sort-Seq Tools website,

`https://github.com/jbkinney/sortseq_tools/`

Methods executed at the command line are shown in yellow boxes, are prefixed by $, and are followed by the resulting standard output (if any). The execution time of each command, as observed on a 2.8 GHz Intel Core i7 MacBook Pro with 16 GB RAM running Python 2.7.9, is shown in brackets below commands and their output. None of these commands took more than 30s to execute. Green boxes are used to illustrate the contents of the input and output text files.

1.1 Overview of Sort-Seq Tools

Sort-Seq Tools provides a suite of command line methods. Barring a few exceptions, each method takes one or more tabular text files as input and returns a tabular text file as output. All input and output files are designed to be human readable. The first line of each tabular text file contains headers describing the contents of each column. All input files are required to have the proper set of columns, which of course depend on the command being executed. By default, input is taken from the standard input and output is written to the standard output. This allows multiple commands to be piped together in series, reducing the need for temporary files. To specify input and output files manually, use the `-i` and `-o` options.

1.2 Simulating data

We begin by simulating a library of variant CRP binding sites. This is accomplished with the following command.

```
$ sortseq_tools simulate_library -w TAATGTGAGTTAGCTCACTCAT -n 100000 -m 0.24 -o library.txt

[9s execution time]
```

Every Sort-Seq Tools method is implemented as a subcommand of `sortseq_tools`. In this example, the operative method is `simulate_library`. The options passed to `simulate_library` are: `-w TAA...CAT`, which specifies the wild type sequence; `-n 100000`, which specifies the number of nonunique mutant sequences to be generated; and `-m 0.24`, which specifies a mutation rate of 24% per position. The output from this command is written to `library.txt` and has the form

```
 ct   seq
 223  TAATGTGAGTTAGCTCACTCAT
  42  TAATGGGAGTTAGCTCACTCAT
  36  TAATGTGAGTTAGCTCACACAT
  35  TAATGTGAGTTAGCTCACTAAT
  33  GAATGTGAGTTAGCTCACTCAT
```

```
 33  TCATGTGAGTTAGCTCACTCAT
 33  TAATGTGAGTTACCTCACTCAT
 32  TAATGTGAGTTAGCTCATTCAT
 32  TAATGTGAGTCAGCTCACTCAT
 ...
```

In `library.txt`, the `ct` column indicates the number of occurrences of each unique sequence. Sequences are sorted by decreasing count. The `seq` header indicates that the sequences listed are DNA sequences. Alternatively, RNA sequences would be indicated by a column header `seq_rna`, and protein sequences would be indicated by a column header `seq_pro`.

To simulate a Sort-Seq experiment, we need a pre-existing model with which to compute the activity of each sequence. Here we use the neighbor model of CRP that was used to simulate the data for Fig. 5 of the main text. This model is represented in a text file, `true_model.txt`, that has the following form:

```
pos      val_AA      val_AC      val_AG      val_AT      val_CA      val_CC    ...
  0     0.081588   -0.019021    0.007188    0.042818   -0.048443   -0.015712 ...
  1     0.033288   -0.005410    0.014198    0.018246   -0.033583   -0.001761 ...
  2    -0.026142    0.008002   -0.029641    0.036698   -0.001028   -0.008025 ...
  3    -0.046159   -0.006071   -0.001542    0.028109   -0.020442   -0.024574 ...
  4    -0.025599   -0.043215   -0.001139   -0.000524   -0.027854   -0.003763 ...
  5    -0.043352    0.014568    0.036066    0.029810    0.038567   -0.029127 ...
  6    -0.007502   -0.018848   -0.018609   -0.016526   -0.043531   -0.075801 ...
  7    -0.010014    0.068860    0.008494    0.070531   -0.016392   -0.064684 ...
  8    -0.012821    0.012587    0.020371   -0.034140    0.004572   -0.025659 ...
...
```

The `pos` column lists the position of each dinucleotide within the binding site, while the columns (`val_AA`, `val_AC`, ...) list the energetic contributions from each of the 16 possible dinucleotides at that position.

To simulate an experiment in which this model of CRP is used to compute the activity of each sequence in the library, we execute the following command.

```
$ sortseq_tools simulate_sort -m true_model.txt -n 4 -i library.txt -o dataset.txt

[30s execution time]
```

The resulting dataset, which is written to `dataset.txt`, has the form,

```
ct    ct_0  ct_1  ct_2  ct_3  ct_4  seq
276     53     1     1    22   199  TAATGTGAGTTAGCTCACTCAT
 55     13     1     1    18    22  TAATGGGAGTTAGCTCACTCAT
 42      6     0     0     0    36  TAATGTGAGTTAGCTCACACAT
 42      7     0     0     1    34  TAATGTGAGTTAGCTCACTAAT
 41      8     0     2     5    26  GAATGTGAGTTAGCTCACTCAT
 38      5     0     0     8    25  TCATGTGAGTTAGCTCACTCAT
 42      9     0     1     7    25  TAATGTGAGTTACCTCACTCAT
 40      8     0     1    13    18  TAATGTGAGTTAGCTCATTCAT
 38      6     0     2     5    25  TAATGTGAGTCAGCTCACTCAT
...
```

Along with the total count for each sequence and the sequence itself, `dataset.txt` lists the counts within the starting library (`ct_0`) and the counts within the 4 sorting bins, arranged from lowest activity to highest activity.

The simulation algorithm used in `simulate_sort` is simple, but it can nevertheless be useful for characterizing Sort-Seq Tools analysis pipelines. It works as follows. First, the model in `true_model.txt` is used to assign an activity to each sequence. Noise is then added to the activity of each sequence, thereby producing a "measurement." Each library sequence is then sorted into one of 4 bins (as specified by the `-n 4` option) based on its measurement. The boundaries of each bin are chosen so that they are equipopulated. The `-sl` flag tells `simulate_sort` to include the original library in the set of output sequences as bin 0. Multiple aspects of this simulation can be modified at the command line by using additional options.

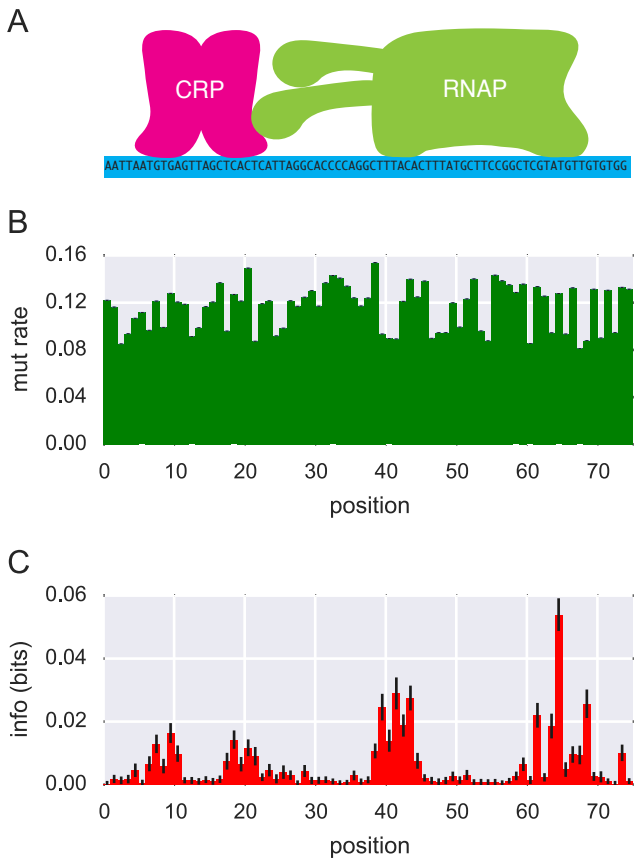

Figure S1: Example mutation and information profiles. (A) The 75 bp region of the *E. coli lac* promoter mutagenized in the full-wt experiment of [1]. Binding sites for CRP and RNAP are indicated. (B) Mutation profile of the full-wt library, indicating an approximately 12% mutation rate across the full 75 bp region. (C) The information profile of the full-wt dataset, which clearly reveals the bipartite binding sites of both CRP and RNAP. Information values were computed using the TPM mutual information estimator; error bars were computed using bootstrap resampling.

1.3 Computing profiles

It is often useful to compute the mutation rate within a set of sequences, e.g., in order to validate the composition of a library. This can be accomplished using the `profile_mut` command as follows

```
$ sortseq_tools profile_mut -b 0 -i dataset.txt -o mutprofile.txt

[3s execution time]
```

Here, the `-b 0` option directs Sort-Seq Tools to extract the library (bin 0) counts from `dataset.txt`. These counts and the corresponding sequences are then used to compute the mutation rate at each position within the sequences; the output file `mutprofile.txt` is

```
 pos    wt   mut
   0     T   0.238331
   1     A   0.237855
   2     A   0.239164
   3     T   0.237935
   4     G   0.236586
   5     T   0.240274
   6     G   0.237895
   7     A   0.239521
   8     G   0.241226
...
```

The `mut` column lists the mutation rate at each position in the sequence, while the `wt` column reports the most frequent base at each position. As expected, the mutation rate at each position is close to the value of 24% specified above; variation from this value reflects the finite number of sequences simulated. An example of a mutation profile computed from real data is shown in Fig. S1B.

To view the frequency of occurrence for every base at each position, use the `profile_freq` command instead:

```
$ sortseq_tools profile_freq -b 0 -i dataset.txt -o freqprofile.txt

[3s execution time]
```

The output file, `freqprofile.txt`, has the form

```
 pos   freq_A     freq_C     freq_G     freq_T
   0   0.077329   0.081255   0.079748   0.761669
   1   0.762145   0.079034   0.078479   0.080343
   2   0.760836   0.081969   0.078201   0.078994
   3   0.081056   0.080620   0.076258   0.762065
   4   0.080144   0.081572   0.763414   0.074870
   5   0.080382   0.078717   0.081175   0.759726
   6   0.078479   0.083158   0.762105   0.076258
   7   0.760479   0.079272   0.081017   0.079232
   8   0.080224   0.082206   0.758774   0.078796
 ...
```

Note that the frequencies listed in each row sum to 1.0, as is required of a list of probabilities.

Information profiles (also called "information footprints" [1]) provide a particularly useful way to identify functional positions within a sequence. These profiles list, for each position in a sequence, the mutual information between a base at that position and the bin in which a sequence having that base is found. Unlike mutation and frequency profiles, which require sequence counts for a single bin only, information profiles are computed from full datasets. The command to do this is

```
$ sortseq_tools profile_info --err -i dataset.txt -o infoprofile.txt

[8s execution time]
```

The output, `infoprofile.txt`, is

```
 pos   info       info_err
   0   0.005211   0.000295
   1   0.007160   0.000273
```

```
   2   0.000688   0.000118
   3   0.024395   0.000621
   4   0.042185   0.000765
   5   0.015530   0.000613
   6   0.073403   0.001098
   7   0.022098   0.000631
   8   0.001993   0.000211
...
```

The `info` column reports the mutual information values in units of bits. The optional `--err` flag causes an additional column to be written, reporting the uncertainty in this mutual information estimate. The specific methods used to compute these information values are described below. An information profile computed from real data is shown in Fig. S1C.

1.4 Quantitative modeling

The command `learn_model` is used to fit models to data. For example, the command

```
$ sortseq_tools learn_model -lm LS -mt MAT -i dataset.txt -o matrix_model.txt

[27s execution time]
```

fits a matrix model (specified by the `-mt MAT` option) to the data in `dataset.txt` using least squares optimization (specified by the `-lm LS` option). The output, written to `matrix_model.txt`, is

```
 pos    val_A       val_C       val_G       val_T
   0    0.894160   -0.904110   -0.545880    0.555830
   1    0.522970   -0.943330   -0.976916    1.397276
   2   -0.006213   -0.546446   -0.311926    0.864585
   3   -0.768769    0.208300   -1.160367    1.720836
   4   -0.801640   -1.745592    2.074855    0.472377
   5   -0.055218   -0.373101   -0.987396    1.415714
   6   -1.329547   -1.678345    2.784233    0.223659
   7    1.495863   -0.153076   -0.791095   -0.551691
   8   -0.789842    0.285597   -0.270838    0.775083
 ...
```

Alternatively, one can learn a neighbor model by instead using the `-mt NBR` option:

```
$ sortseq_tools learn_model -lm LS -mt NBR -i dataset.txt -o neighbor_model.txt

[30s execution time]
```

The `-lm IM` option specifies parameter inference using information maximization, while the `-lm ER` option specifies the use of enrichment ratio calculations (requires `-mt MAT`).

1.5 Evaluating models

At the end of the day, the purpose of having a quantitative model is to be able to predict the activity of arbitrary sequences. This basic operation is accomplished using the `evaluate_model` command. For instance, to evaluate the inferred matrix model, `matrix_model.txt`, on the sequences in `dataset.txt`, use the following command:

```
$ sortseq_tools evaluate_model -m matrix_model.txt -i dataset.txt -o dataset_with_values.txt

[20s evaluation time]
```

The output file, `dataset_with_vals.txt`, contains the same information as `dataset.txt`, along with an additional column `vals` listing the values predicted by the model:

```
 ct    ct_0  ct_1  ct_2  ct_3  ct_4  val         seq
 276     53     1     1    22   199  17.159008   TAATGTGAGTTAGCTCACTCAT
  55     13     1     1    18    22  14.755898   TAATGGGAGTTAGCTCACTCAT
  42      6     0     0     0    36  20.389269   TAATGTGAGTTAGCTCACACAT
  42      7     0     0     1    34  18.348347   TAATGTGAGTTAGCTCACTAAT
```

```
  41     8     0     2     5    26  16.057298  GAATGTGAGTTAGCTCACTCAT
  38     5     0     0     8    25  15.692708  TCATGTGAGTTAGCTCACTCAT
  42     9     0     1     7    25  16.691966  TAATGTGAGTTACCTCACTCAT
  40     8     0     1    13    18  15.352424  TAATGTGAGTTAGCTCATTCAT
  38     6     0     2     5    25  16.681243  TAATGTGAGTCAGCTCACTCAT
 ...
```

Often, it is useful to scan a model over all sequences embedded within larger contigs. To do this, Sort-Seq Tools provides the command `scan_model`, which is implemented as follows

```
$ sortseq_tools scan_model -n 100 -m matrix_model.txt -i genome_ecoli.fa -o genome_sites.txt

[31s evaluation time]
```

Here `genome_ecoli.fa` is a FASTA file containing the entire genome of *Escherichia coli*, and the option `-n 100` specifies that only the 100 top-scoring sites are to be recorded. The output file, `genome_sites.txt`, has the form

```
val         seq                      left      right     ori  contig
25.994723   TTTTGTGAACTATATCACAATT   4273191   4273212     +  MG1655.fa
25.502928   ATGTGTGATCGTCATCACAATT    141283    141304     +  MG1655.fa
25.026216   ATTTGTGATCTGGATCGCGTTT   1614957   1614978     -  MG1655.fa
24.934751   ATATGTGATCTGAATCTCATTA   2229787   2229808     -  MG1655.fa
24.883642   TTTTGTGATCAATTTCAAAATA   4099532   4099553     +  MG1655.fa
24.285880   AATCGTGATTTACATCACAATT   1528510   1528531     -  MG1655.fa
24.274854   TTCTGTGATTGGTATCACATTT     42067     42088     +  MG1655.fa
24.249231   ATATGTGATTCATATCACATAT   4589511   4589532     +  MG1655.fa
24.184566   TTATGTGATAAAAGTCACATTT   2031480   2031501     +  MG1655.fa
 ...
```

which lists the value assigned to each sequence (`val` column), the identified sequence (`seq` column), the left-most and right-most positions within the sequence (`left` and `right` columns), the orientation of the sequence (`ori` column; '+' = top strand, '-' = bottom strand), and the FASTA file contig containing the site.

A good way to assess the quality of a model is to compute its predictive information on a massively parallel data set. This can be done using the `predictiveinfo` command:

```
$ sortseq_tools predictiveinfo -m matrix_model.txt -ds dataset.txt
info
0.569086

[12s evaluation time]
```

The output from this command reports the predictive information in units of bits. The `-mt MAT` option signals that the model is a matrix model. To run a neighbor model over this dataset, use the `-mt NBR` option:

```
$ sortseq_tools predictiveinfo -m true_model.txt -ds dataset.txt
info
0.724236

[14s evaluation time]
```

2 Comparison to dms_tools

As described in the main text, dms_tools [2] provides routines for computing enrichment ratios, which are often interpreted as matrix models describing sequence-function relationships [3]. These enrichment ratios can be computed in two different ways: using the analytic formula described in Eqs. 4 and 5 of the main text (which can be derived as a maximum likelihood estimator), and through Monte Carlo sampling of a Bayesian posterior distribution.

Fig. S2 compares the performance of matrix models inferred using the enrichment ratio (ER) method provided by Sort-Seq Tools to both the dms_tools analytic (DTA) models and the dms_tools Monte Carlo

(DTM) models. This comparison was performed on all experimental data analyzed in the main text, and on these data sets we find these three modeling methods to be virtually indistinguishable. These results make sense. The ER and DTA inference methods are mathematically identical, and should produce output that is identical modulo minor differences due to variation in numerical implementation. As shown in Figs. S3A and S3C, this is indeed what is found. The DTM inference method, by contrast is very different in kind than DTA. But because DTM is a Bayesian computation, it is expected to provide virtually the same results as DTA in the large data limit. Again this is what we find (Figs. S3B and S3D), supporting the conclusion that this limit is the relevant one for all of the data sets analyzed in this paper.

3 Mutual information calculations

3.1 Information profiles

To construct information profiles, Sort-Seq Tools must estimate (at each sequence position) the mutual information between the character occurring at that position and corresponding sequence bin. In terms of the joint probability distribution $p_l(M, C)$ relating bin M to the character C at position l, this mutual information value is given by

$$I_l[M;C] = \sum_{M,C} p_l(M,C) \log_2 \left[\frac{p_l(M,C)}{p_l(M)p_l(C)} \right]. \tag{1}$$

In this formula, the probability distribution $p_l(M, C)$ is to be interpreted as what one would obtain if the dataset consisted of an infinite number of independent measurements. In practice, however, one has only a finite number of measurements. The issue of how to best estimate $I_l[M, C]$ from a limited amount of data is nontrivial [1]. Sort-Seq Tools allows the user to compute information profiles using three different mutual information estimators: the "naive" estimator, the "TPM" estimator proposed by Treves and Panzeri [6] following the work of Miller, [7], and the "NSB" estimator proposed by Nemenman, Shafee, and Bialek [8]; see also [9].

The simplest estimator is the naive estimator, which is calculated by replacing the probability distribution $p_l(M, C)$ Eq. (1) with the set of frequencies f_{Cl}^M, described in Eq. 5 of the main text (with pseudocount $\lambda = 0$), i.e.

$$I_l^{\text{naive}}[M;C] = \sum_{M,c} f_{cl}^M \log_2 \left[\frac{f_{cl}^M}{\left(\sum_c f_{cl}^M\right)\left(\sum_M f_{cl}^M\right)} \right] \tag{2}$$

The quantity I_{naive} is known to overestimate mutual information. When substantial counts exist for all possible values of C and M one can analytically compute the order N^{-1} correction to mutual information. This gives the TPM estimator:

$$I_l^{\text{TPM}}[M;C] = I_l^{\text{naive}}[M;C] - \frac{(n_M - 1)(n_C - 1)\log_2 e}{2N}. \tag{3}$$

where n_M and n_C respectively denote the number of bins and characters. Alternatively, when the probability distribution $p(M, C)$ is not well-sampled, mutual information can be computed using the NSB estimator. Computing $I_l^{\text{NSB}}[M;C]$ requires computing a one-dimensional integral and is more computationally expensive than the other two approaches.

Bootstrap resampling is used to estimate uncertainties in both I^{naive} and I^{TPM}. Uncertainties in I^{NSB}, by contrast, are computed analytically as described in [9].

3.2 Predictive information

Activity predictions of the linear models supported by Sort-Seq Tools are continuous real numbers. This necessitates a different procedure for estimating predictive information $I[M;R]$ – the mutual information

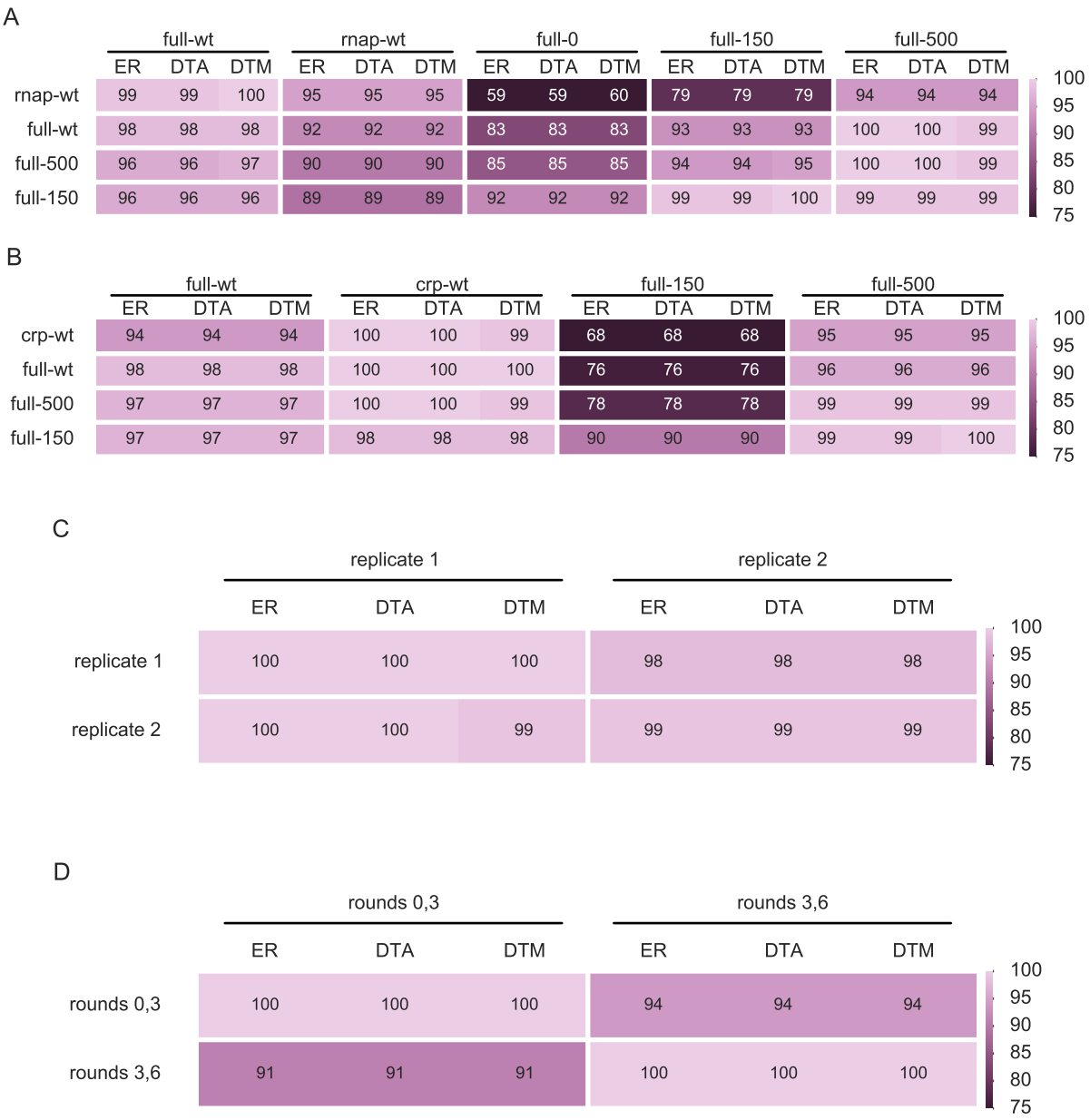

Figure S2: Performance comparison of models inferred using Sort-Seq Tools and dms_tools. Results for three different matrix model inference methods are shown: the analytic enrichment ratio calculations of Sort-Seq Tools (ER), the analytic enrichment ratio calculations of dms_tools (DTA), and the Monte Carlo enrichment ratio calculations of dms_tools (DTM). Comparisons are performed on (A) RNAP Sort-Seq data from [1], (B) CRP Sort-Seq data from [1], (C) MPRA data from [4], and (D) DMS data from [5], as respectively described in Figs. 4A, 4B, 6A, and 6B of the main text. Note that, unlike Sort-Seq Tools, dms_tools does not support the inference of neighbor models, nor does it support the use of least squares optimization or information maximization as a means of parameter inference.

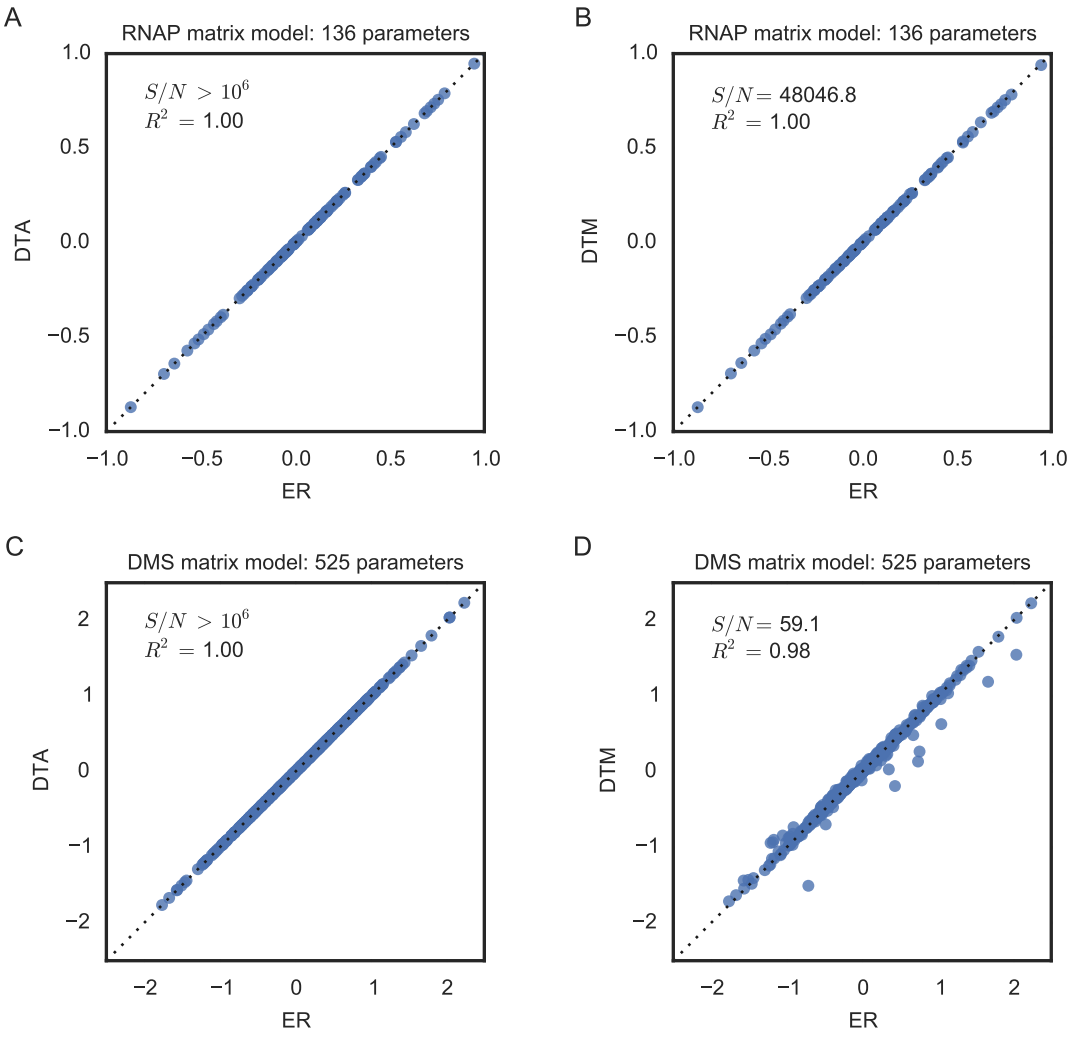

Figure S3: Parameter comparison of models inferred using Sort-Seq Tools and dms_tools. The parameters of models fit to (A,B) the RNAP binding site in the full-wt data of [1] and (C,D) the DMS data from rounds 0,3 of [5]. ER, DTA, and DTM are defined as in Fig. S2.

between model prediction R and sequence bin M – than was described above for estimating $I[M;C]$. Sort-Seq Tools implements a kernel density estimation procedure similar to the method described in [1], but modified for increased speed.

First each model prediction R is discretized with resolution equal to 1% of the standard deviation of all model predictions in the population. Next, model prediction values are sorted and replaced with their corresponding rank order. Rank order values are then further discretized so as to occupy 1000 equipopulated bins. These binned values are then convolved with a Gaussian kernel having a standard deviation of 20 bin widths. This procedure provides an estimated continuous distribution, $p(R, M)$, which is then used in Eq. 10 of the main text to estimate predictive information.

To estimate uncertainties in $I[M;R]$, this same information estimation procedure is carried out multiple times using different 50% subsets of the data. The uncertainty in $I[M;R]$ is then computed as

$$\delta I[M;R] = \sqrt{\frac{\mathrm{var}(I^{50\%}[M;R])}{2}} \tag{4}$$

where $I^{50\%}[M;R]$ denotes the information estimates computed using the 50% subsamples.

References

[1] Kinney, J.B., Murugan, A., Callan, C.G., Cox, E.C.: Using deep sequencing to characterize the biophysical mechanism of a transcriptional regulatory sequence. Proc Natl Acad Sci USA **107**(20), 9158–9163 (2010)

[2] Bloom, J.D.: Software for the analysis and visualization of deep mutational scanning data. BMC Bioinformatics **16**, 168 (2015)

[3] Stormo, G.D.: DNA binding sites: representation and discovery. Bioinformatics **16**(1), 16–23 (2000)

[4] Melnikov, A., Murugan, A., Zhang, X., Tesileanu, T., Wang, L., Rogov, P., Feizi, S., Gnirke, A., Callan, C.G., Kinney, J.B., Kellis, M., Lander, E.S., Mikkelsen, T.S.: Systematic dissection and optimization of inducible enhancers in human cells using a massively parallel reporter assay. Nat Biotechnol **30**(3), 271–277 (2012)

[5] Fowler, D.M., Araya, C.L., Fleishman, S.J., Kellogg, E.H., Stephany, J.J., Baker, D., Fields, S.: High-resolution mapping of protein sequence-function relationships. Nat Methods **7**(9), 741–746 (2010)

[6] Treves, A., Panzeri, S.: The upward bias in measures of information derived from limited data samples. Neural Comput **7**(2), 399–407 (1995)

[7] Miller, A.G.: Note on the bias of information estimates. Inform Theory Psychol Prob Meth **2**, 95–100 (1955)

[8] Nemenman, I., Shafee, F., Bialek, W.: Entropy and Inference, Revisited. In: Dietterich, T.G., Becker, S., Ghahramani, Z. (eds.) Adv Neural Inf Proc Syst. MIT Press, Cambridge, MA (2002)

[9] Kinney, J.B.: Biophysical models of transcriptional regulation from sequence data. PhD thesis, Princeton University (September 2008)