Supporting File

1 Correlation distributions of synthetic data

We intended to simulate the synthetic datasets with both positive and negative feature correlation. To confirm, we randomly selected nine simulated datasets for investigating. For each dataset, we computed the empirical correlation matrix and plotted the pairwise feature correlation histogram, shown in Supporting Figure 1.

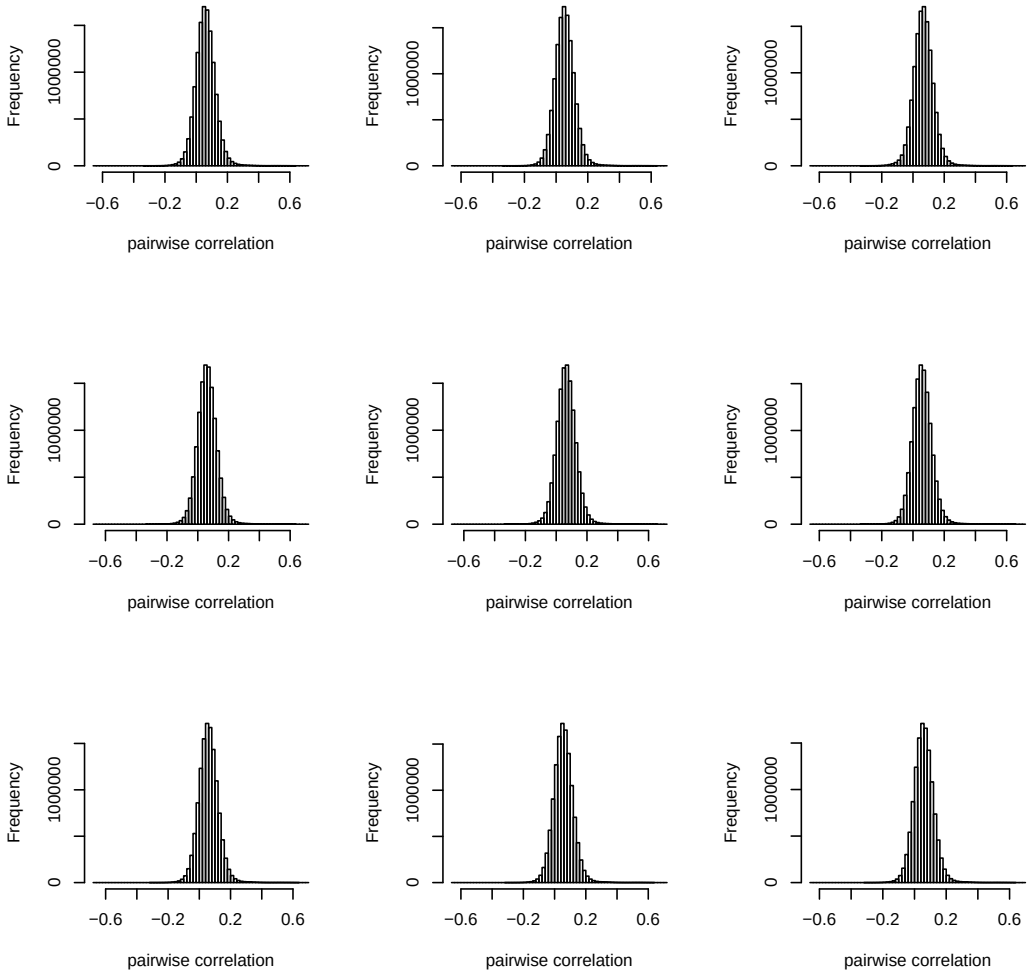

Figure 1: Histograms of the pairwise feature correlation distributions for randomly selected simulation datasets.

2 Sensitivity analysis

2.1 Initial values

The initial values of the neural network part in forgeNet are generated by the deep learning library (i.e. Tensorflow). To test the reproducibility of forgeNet, we used one dataset for each simulation case and ran 10 repeated experiments. The testing results of forgeNet-RF and forgeNet-GBM are shown in Supporting Table 1.

forgeNet-RF	#true predictors				
	15	30	45	60	75
1	0.638	0.822	0.709	0.786	0.772
2	0.609	0.817	0.663	0.798	0.792
3	0.556	0.809	0.721	0.79	0.778
4	0.65	0.817	0.701	0.776	0.791
5	0.624	0.839	0.673	0.778	0.774
6	0.596	0.834	0.725	0.792	0.763
7	0.627	0.816	0.7	0.788	0.791
8	0.567	0.812	0.7	0.784	0.787
9	0.637	0.838	0.722	0.79	0.783
10	0.645	0.824	0.732	0.8	0.755
Avg.	0.615	0.823	0.705	0.788	0.778
S.d.	0.033	0.011	0.023	0.008	0.013

forgeNet-GBM	#true predictors				
	15	30	45	60	75
1	0.602	0.798	0.687	0.791	0.771
2	0.606	0.8	0.733	0.775	0.773
3	0.567	0.809	0.713	0.79	0.723
4	0.552	0.789	0.652	0.77	0.737
5	0.587	0.786	0.651	0.773	0.755
6	0.631	0.791	0.667	0.784	0.741
7	0.645	0.791	0.648	0.79	0.73
8	0.618	0.803	0.662	0.759	0.777
9	0.657	0.827	0.671	0.768	0.748
10	0.635	0.805	0.701	0.778	0.743
Avg.	0.61	0.8	0.679	0.778	0.75
S.d.	0.034	0.012	0.029	0.011	0.019

Table 1: Testing ROC-AUC of repeated experiments for fixed datasets. Left: forgeNet-RF. Right: forgeNet-GBM.

2.2 Hyper-parameters

In this subsection, we demonstrate an example of how we choose the hyper-parameters using grid search. We divided the hyper-parameters into two sets: network architecture-related parameters, and training-related parameters. The former include the number of hidden layers and the number of hidden neurons in each layer, and the latter include the dropout proportion, the learning rate and the batch size.

In forgeNet, the first hidden layer (graph-embedded layer) always has the same amount of hidden neurons as the number of features p, hence we only need to tune subsequent hidden layers. We set a feasible space with 2, 3 and 4 hidden layers (including the first hidden layer), as the neural network could not be too deep in our small sample scenarios. For the number of hidden neurons, we used the conventional choices such as 128, 64, 32 and 16. Again, to avoid overfitting, hidden neurons over 128 were not considered. For training related parameters, we designed several representative options and tested their combinations. A hyper-parameter tuning example is shown in Supporting Table 2. In this particular example, the best neural network structure was with three hidden layers, and the two fully connected layers had 64 and 16 neurons respectively. Also, the best dropout proportion/learning rate/batch size combination was 0.5/0.0001/32.

Dropout/Learning rate/Batch size	#hidden layers & #hidden neurons					
	p+64	p+128	p+64+16	p+128+32	p+128+32+16	p+128+64+16
0.2/0.0001/8	0.79	0.802	0.8	0.767	0.817	0.795
0.2/0.0001/16	0.821	0.772	0.798	0.762	0.814	0.81
0.2/0.0001/32	0.804	0.806	0.781	0.781	0.815	0.795
0.2/0.001/8	0.735	0.766	0.719	0.746	0.739	0.747
0.2/0.001/16	0.772	0.718	0.7	0.756	0.728	0.738
0.2/0.001/32	0.739	0.764	0.737	0.721	0.768	0.768
0.2/0.01/8	0.734	0.735	0.756	0.729	0.722	0.725
0.2/0.01/16	0.758	0.679	0.702	0.724	0.712	0.71
0.2/0.01/32	0.782	0.696	0.711	0.747	0.729	0.717
0.5/0.0001/8	0.799	0.799	0.807	0.794	0.804	0.787
0.5/0.0001/16	0.797	0.811	0.809	0.787	0.805	0.779
0.5/0.0001/32	0.803	0.824	**0.837**	0.799	0.801	0.797
0.5/0.001/8	0.739	0.739	0.76	0.735	0.734	0.736
0.5/0.001/16	0.761	0.772	0.777	0.761	0.754	0.737
0.5/0.001/32	0.775	0.779	0.781	0.787	0.799	0.78
0.5/0.01/8	0.708	0.697	0.692	0.716	0.675	0.749
0.5/0.01/16	0.657	0.746	0.735	0.747	0.725	0.724
0.5/0.01/32	0.735	0.735	0.718	0.72	0.774	0.706
0.8/0.0001/8	0.804	0.794	0.789	0.799	0.816	0.783
0.8/0.0001/16	0.804	0.801	0.826	0.812	0.81	0.774
0.8/0.0001/32	0.787	0.782	0.794	0.797	0.816	0.816
0.8/0.001/8	0.762	0.739	0.818	0.812	0.783	0.807
0.8/0.001/16	0.781	0.764	0.802	0.783	0.806	0.793
0.8/0.001/32	0.808	0.777	0.819	0.799	0.786	0.79
0.8/0.01/8	0.727	0.699	0.781	0.782	0.51	0.52
0.8/0.01/16	0.731	0.711	0.781	0.764	0.79	0.525
0.8/0.01/32	0.736	0.733	0.783	0.743	0.797	0.65

Table 2: An example of hyper-parameter tuning for forgeNet (RF). The column names denote **hidden layers** and their corresponding numbers of hidden neurons. For example, "p+64+16" stands for a neural network architecture with a p-dimensional input layer, a p-dimensional graph-embedded layer, a 64-dimensional fully connected hidden layer, a 16-dimensional fully connected hidden layer and a two-dimensional output layer.

3 Computational cost

In this section, we examine the scalability of the forgeNet model by using synthetic datasets with large sample sizes and extremely large feature spaces. We set candidate sample sizes as 400, 800, 2000 and 5000 and candidate number of features 5000, 10000, 20000 and 50000. The numbers should cover the upper size limit of a typical trancriptomic dataset well. We compared both the time cost and the memory cost of forgeNets with their tree-ensemble counterparts. The experiments were run on a workstation with dual Xeon Gold 6136 processors, 256 GB RAM, and a single Nvidia Quadro P6000 GPU.

The results for forgeNet-RF are shown in Table 3 and 4, and the results for forgeNet-GBM are shown in Table 5 and 4. Detailed reading instruction can be found in corresponding table captions. From the tables, it was not surprising to see that forgeNets took more time and used more space than their tree feature extractors alone, since forgeNets bore additional deep neural network computation. However, the extra time and memory (GPU) induced by forgeNets were well acceptable, indicating the plausibility of the method for large-scale data.

Time (sec)	#features			
#samples	5000	10000	20000	50000
400	4.5/14.3	4.9/15.5	5.3/18.7	5.6/24.0
800	5.3/19.7	5.4/24.0	5.7/26.3	6.3/31.9
2000	5.9/34.4	6.4/39.7	7.1/45.6	9.3/56.2
5000	7.5/68.8	9/81.4	11.6/88.3	15.6/108.6

Table 3: Computational time for forgeNet-RF and the corresponding RF model alone. The time used by RF and by forgeNet-RF are separated by "/". For example, "4.5/14.3" means the time of running RF is 4.5 seconds while running the entire forgeNet takes 14.3 seconds.

Memory (MB)	#features			
#samples	5000	10000	20000	50000
400	141.2 (73.1)	171.8 (98.6)	233.2 (145.6)	417.2 (234.6)
800	171.7 (91.4)	232.9 (173.1)	355.3 (203.2)	722.4 (292.0)
2000	263.3 (135.6)	416.0 (181.9)	721.5 (271.2)	1637.9 (352.2)
5000	492.2 (137.9)	873.8 (221.6)	1637.1 (266.9)	3926.8 (381.3)

Table 4: Memory usage for forgeNet-RF and the corresponding RF model alone. Since we used the GPU version of the Tensorflow deep learning library, forgeNet merely induced additional space cost in terms of GPU memory only, and the RAM usage remained the same as the corresponding tree-ensemble method. The extra (GPU) memory usage of forgeNet-RF is shown in a bracket. For example, "141.2 (73.1)" means the RF extractor used 141.2 MB RAM memory, and to train the entire forgeNet, 73.1 MB extra GPU memory were also used.

Time (sec)	#features			
#samples	5000	10000	20000	50000
400	5.5/11.0	8.4/14.4	11.8/17.9	26/32.7
800	8.3/17.9	13.3/23.5	23.7/33.6	49.9/59.9
2000	20.4/41.7	38.4/59.3	68.3/89.8	143/159.3
5000	57.1/104.9	114.5/177.1	190.7/237.4	539/604.9

Table 5: Computational time for forgeNet-GBM and the corresponding GBM model alone. The time used by GBM and by forgeNet-GBM are separated by "/". For example, "5.5/11.0" means the time of running GBM is 5.5 seconds while running the entire forgeNet takes 11.0 seconds.

Memory (MB)	#features			
#samples	5000	10000	20000	50000
400	138.4 (2.4)	169.4 (4.5)	231.3 (2.2)	417 (2.8)
800	168.9 (6.5)	230.4 (8.7)	353.4 (10.1)	722.2 (10.9)
2000	260.5 (25.2)	413.5 (44.4)	719.6 (38.6)	1637.7 (21.9)
5000	489.5 (70.5)	871.4 (145.7)	1635.2 (72.9)	3926.6 (166.6)

Table 6: Memory usage for forgeNet-GBM and the corresponding GBM model alone. Since we used the GPU version of the Tensorflow deep learning library, forgeNet merely induced additional space cost in terms of GPU memory only, and the RAM usage remained the same as the corresponding tree-ensemble method. The extra (GPU) memory usage of forgeNet-GBM is shown in a bracket. For example, "138.4 (2.4)" means the GBM extractor used 138.4 MB RAM memory, and to train the entire forgeNet, 2.4 MB extra GPU memory were also used.