1 Simulation Experiments

1.1 Experiment Design

1.1.1 The First Experiment

In the first simulation experiment, we want to see how the parameter ρ influences the performance of *HyperGene*. In the optimization framework, ρ is a parameter to control how much weight we put on the prior knowledge. If there is no prior knowledge, it is just a restriction on the variance of weights for all hyperedges. If ρ is large, the variance of weights for hyperedges will be small, which means they will get similar weights in our algorithm. In this case, *HyperGene* will perform in a similar way as what the original hypergraph-based algorithm does, because the original hypergraph-based algorithm treats them as the same important features. However, if ρ is small, *HyperGene* will make the variance large so we can expect an improvement of classification performance based on the new weights. In this case, we do believe some of the features are much more important than others and we want to assign them higher weights than average.

To verify these assumptions, we test it on randomly generated dataset. We use the same number of samples (training and test) and features as the real data to best keep the property of the original dataset. In *HyperGene*, features are modeled as hyperedges in the hypergraph. In the simulation, we use the following model to generate them. Each of the hyperedges is in one of the two groups: The first group is the set of hyperedges which are useful in classification of test samples, while the second group is the set of hyperedges which are useless in helping making a prediction. We use mutual information as the measurement of how useful a hyperedge is to predict the labels of test samples. So when we generate hyperedges in the first group, we make the mutual information of each hyperedge and the label be around some positive constant. When we generate hyperedges in the second group, we don't use the label information and just generate hyperedges randomly, then each hyperedge is independent to label thus their mutual information is approximately zero. (Because we allow certain randomness in generating hyperedges, the mutual information cannot be restricted to be a fix constant.)

1.1.2 The Second Experiment

In the second experiment, we want to see how well *HyperGene* can use prior knowledge to improve the performance. Instead of generating hyperedges in the first group independently as in the first experiment, we use another model to generate them in this experiment. First we generate a hyperedge with very

Table 1
Effect of prior knowledge on performance

SVM(linear)	SVM(rbf)	Hyper	HyperGene+No Prior	HyperGene+Prior1	HyperGene+Prior2
		$\alpha = 0.3$	$\alpha = 0.3, \rho = 0.001$	$\alpha = 0.7, \rho = 0.1$	$\alpha = 0.7, \rho = 0.1$
0.6318	0.6391	0.7164	0.8217	0.9310	0.9237

HyperGene+No Prior: We use no prior knowledge.
HyperGene+Prior1: We use only correct prior knowledge.
HyperGene+Prior2: We also add false information to prior knowledge.

high mutual information to the label. Then we use 10 hyperedges to replace this highly predictive hyperedge in such a way that the 10 hyperedges will equally cover the samples which are covered by the highly predictive one. (Then the highly predictive hyperedge is removed.) The generating of hyperedges in the second group is the same as in the first experiment.

Now we have the prior knowledge that the 10 hyperedges are actually from one improtant hyperedge, so we want to assign similar weights to the 10 hyepredges. To construct a graph laplacian with such prior knowledge, first we denote each feature as a vertex in a simple graph and the weight of each edge is the pairwise relation of the corresponding two features. Then we construct the graph laplacian to make the relations of the 10 hyperedges in the first group much stronger than those of other pairs. In other words, we want to make the 10 hyperedges a strongly connected clique in the graph. We also add some noise to the graph laplacian to see how well *HyperGene* can deal with the case that there is false information in the prior knowledge. To add such noise, we randomly select 10 hyperedges in the second group and also make them a strongly connected clique in the graph. Same as the first experiment, we randomly generate 100 datasets and take the average ROC score for each combination of parameters (α=0.3, 0.5, 0.7 and $\rho = 0.001, 0.01, 0.1$). We report the best result and the corresponding parameters for each algorithm.

1.2 Experiment Results

In the experiment, we tried different combinations of the two parameters α and ρ in *HyperGene*. We also set another parameter *percentage* in the experiment, which is used to control how many useful hyperedges (hyperedges in the first group) we want to generate in the simulated dataset. Because in real biomedical datasets, most of features (gene expression level, SNP, ...) are noise, we set *percentage* to be very low to accord with this fact. To well evaluate the performance of *HyperGene*, we also compared it with Support Vector Machine. For each combination of the three parameters, we randomly generate 100 datasets and report the average ROC scores for all algorithms.

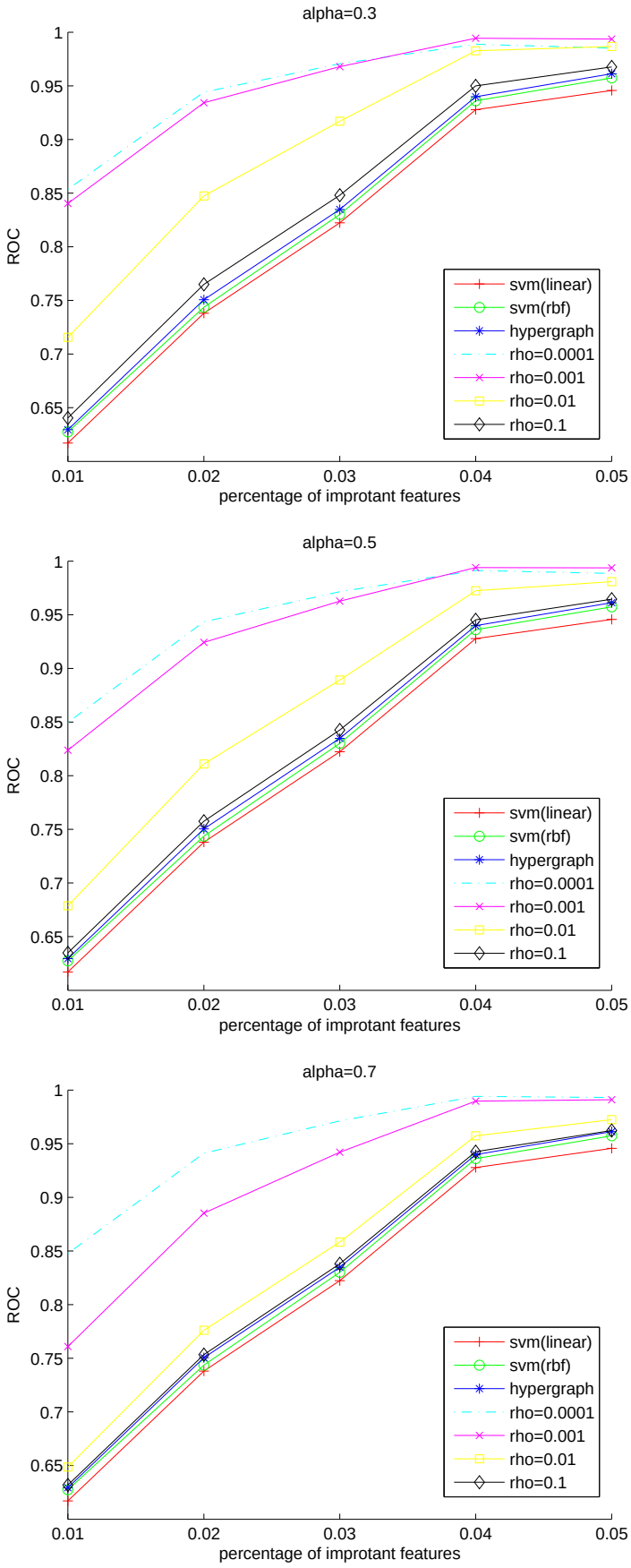

Fig. 1. Comparison of ROC score on simulated data. In the three figures, α is set to 0.3, 0.5, 0.7, and ρ is set to 0.1, 0.01, 0.001, 0.0001 for each α.

The standard deviation of *HyperGene* is slightly smaller than SVM. (Because of lack of space, we don't report stand deviation in the paper). From Fig 1, we can see if the percentage of important feature is really small in the dataset, using a smaller ρ will improve the performance of *HyperGene*. However, if the number of important features is really large (In this experiment, it is 5% of total number of features. The number will vary if we use a different mutual information threshold to generate important features) that nearly all classification algorithm can have a very good performance (such as ROC$\geq$0.95), the function of ρ is not important anymore. And in this case, we can observe smaller rho starts to work worse than larger ρ. It is understandable because if there are enough good features, it is a better choice to use a larger ρ to restrict the variance of weights so more edges can have higher weights.

From Table 1, we can see although *HyperGene* already performances better than SVM and original hypergraph-based algorithm without prior knowledge, by introducing prior knowledge, *HyperGene* will further improve its performance. Larger ρ works better this time because we have prior knowledge, which indicates if we trust more on correct prior knowledge, we will get better performance. Another thing worth noting is that even after we add some false information to the prior knowledge, the performance of *HyperGene* doesn't change much, which shows *HyperGene* has some tolerance to noise. Even if the prior knowledge is not totally correct, *HyperGene* can still use the information provided to improve the performance.