Guangju Wang
Math 6380
March 13, 2017

Mini-Project 1 Report

gene set selection based on linear regression model

There are 149 samples, three features set and other complementary information in this problem. The first problem is where to start. Based on the domain knowledge from high school biology and information provided by Prof. Yao, I started with the RNA feature set. In terms of programming language, I used python. In python, I can easily execute large scale computation on remote server, which has higher computation power.

To start with, I used pandas 'dataframe' to parse the excel file into separated feature sets. And further separated the available training set from the test, in the following way.

```
data = pd.ExcelFile('data_cellline1.xlsx')
```

```
#data cleanup
ic50 = data.parse(0).transpose()

rna = data.parse(1).transpose()[1:]
cnv = data.parse(2).transpose()[1:]
mutation = data.parse(3).transpose()[1:]
ic50 = ic50.set_index(0)
train_list = ic50[2]!= '?'
test_list = ic50[2]== '?'
```

One observation is that pandas 'dataframe' is efficient in manipulating large scale of data, but slow in parsing them from excel file, in terms of computation time. Parsing work takes tens of seconds but the manipulation(transpose, duplicate) takes much less.

At first I though my work is to predict all three of 'ic50-24hrs', 'ic50-40hrs' and 'ic50-72hrs'. So I checked the correlation between them. No surprise, they have strong positive

correlation. That means, if a tissue has high response in the first 24 hours, it it very likely that it has high response level after 72 hours.

```
#check the correlation between the three responses
print corr(ic50_train[2], ic50_train[3])
print corr(ic50_train[2], ic50_train[4])
print corr(ic50_train[3], ic50_train[4])
#the crucial task is to get a good predicter of ic50-24, whi
```

```
(0.53628491307287618, 5.7506838202445851e-11)
(0.42407962657898152, 5.4950735453706101e-07)
(0.63886296277099752, 3.7553742971781563e-16)
```

Speaking of parameter selection, my first thought was to use lasso. But I had no idea how lasso performs on scale of this large. So I tried Cross-Validation Lasso from 'scikit-learn' and the algorithm returns a alpha of around 26 and a set of for parameters.

```
#define a cross validation function to find the best lasso parameter and th
#What defines best? Try several lasso parameters and select the one that gi
#Use scikit-learn lassoCV as an example
lassocv24 = lassoCV(alphas= np.arange(20, 30, 0.02))
lassocv24.fit(rna_train, ic50_train[2])
#lassocv for response after 24 hours
# Method replaced by multilassoCV
```

The four selected parameters are: 7595, 8995, 9415, 14674 from the RNA feature set. Multi-task Lasso cross-validation algorithm returns the same set for all three responses. Although lasso, in terms of parameters, does not necessary give us the best result. But with the belief that this result will not be too worse off, I proceed with the four parameters and run some validation.

The lasso result is like a 'local- optimal', meaning that there might be another feature set that gives us a better result but the lasso cross-validation algorithm will never find it. Then with my limited CS knowledge, I use a similar method to simulated-annealing: I randomly add another parameter to the feature set and then use cross validation MSE to validate if it is a better result or not. Here, another observation is that I found scikit-learn, also powerful, did not provide a efficient way to compute the cross-validation MSE of you model. So I defined a function 'msetest' that gives me the cross-validated MSE of a specific feature set, on a linear regression model basis.

```
#test the average cv MSE of the 7 parameters
def msetest(x, y, loc_param, split_ratio = 0.8, maxiter = 1000):
    simplelr = lr()
    mselist = []
    for _ in range(maxiter):
        temp_train_set = np.random.rand(x.shape[0]) < 0.8
        simplelr.fit(x[loc_param][temp_train_set], y[temp_train_set])
        mselist.append(mse(simplelr.predict(x[loc_param][~temp_train_set]),
    return np.average(mselist)
```

```
# selected 1304 as the new feature to enter the set, run another round
# add a few more features so that we can run another lasso regression
# we have to define a function for this
def add_new_parameters(x, y, loc_param, startingMSE, split_ratio = 0.8, max
    simple_lr = lr()
    training_set_result = []
    i = 0
    while i <maxiter:
        i = i + 1
        for _ in list(x):
            resultMSE = msetest(x, y, loc_param)
            if _ not in loc_param and np.random.random() < 1:
                if msetest(x, y ,loc_param + [_]) < resultMSE:
                    loc_param = loc_param + [_]
    #     return 'a better MSE of', resultMSE, 'with parameters set', loc_p
        print i
    return loc_param
```

At first I didn't use cross validation to validate the new feature set. As long as the feature set has a better MSE on one random, specific train-test pair, it is added to the feature set. After several rounds of parameter selection. I ended up with another set of features that seems to give a better result but with a higher dimension. The new set has 17 features which apparently too large. My next concern, apparently, is that this may induce overfitting.

Not surprisingly, using all 17 features will give a worse cross validate MSE. Another round of cross validation lasso gives a smaller feature set. But compared to the original feature set, this time, it kept three other parameters: 2593, 2802, and 11656.

To validate this new feature set, I run another round of cross-validation. This time, this feature of length 7 gave a better cross-validated MSE compared to the original feature set:

```
In [67]: msetest(rna_train, ic50_train[4], [7595, 8995, 9415, 14674])
Out[67]: 0.95191314360091817
```

```
In [66]: test(rna_train, ic50_train[4], [7595, 8995, 9415, 14674, 2593, 2802, 11656])
Out[66]: 0.88414402372224687
```

On kaggle.com I submitted the linear regression model with this feature set. The result was improved.

My last attempt was to work on the training set. When I run MSE test on different training set, some training - validation pair has very good performance on MSE. I though that due to the variability and randomness of the sampled data, it might be true that some subset of samples will give me a better fitted model than using all the samples. I used another small randomized algorithm to select a better training set.

```
def find_best_training_set(x, y, loc_param, split_ratio = 0.8, maxiter = 10
    simplelr = lr()
    bestmse = 1
    train_set_list = []
    for _ in range(maxiter):
        temp_train_set = np.random.rand(x.shape[0]) < 0.8
        simplelr.fit(x[loc_param][temp_train_set], y[temp_train_set])
        a = mse(simplelr.predict(x[loc_param][~temp_train_set]), ic50_train
        if a < bestmse:
            bestmse = a
            train_set_list.append(temp_train_set)
    print bestmse
    return train_set_list
```

Then I fitted another model, with the same feature set, but a smaller training set. However, the result from kaggle.com indicated that this did not improve the MSE on the test set.

This summarizes my work on mini-project 1. The notebook is also attached.