MATH6380J Mini Project 2 Report

Yue JIANG (20238316)
Zhenzhen LI (20303719)
Lizhang MIAO (20296174)

April 10, 2017

1 Background and Data Description

The one-drug dataset provided by Prof. Jiguang Wang and Dr. Biaobin Jiang contains 642 cancer cell line samples in response to one drug, with 60 binary features as gene mutation status. The response variable that we want to predict is drug sensitivity, which is measured by logarithmic IC50. IC50 values are the dosage amounts of the drug such that after a period of time the experimental cancer cell lines are killed by 50%.
A random subset of 100 cell line responses are withdrawn to the test sample. Among the remaining 542 samples, we randomly choose 500 from them as the training sample and the 42 samples left are regarded as the validation sample.

2 Methodology

2.1 Elastic Net

As we have 60 explaining variables in total, we want to minimize the prediction error as well as obtaining sparsity. And the resulting sparse variables vector can be regarded as the most important features that affect IC50s. Elastic net is a technique that uses both L_1 and L_2 penalty. The estimates from elastic net are defined by

$$\hat{\beta} = argmin_\beta(\frac{1}{2N}\sum_{i=1}^{N}(y_i - x_i^T\beta)^2 + \lambda((1-\alpha)\frac{1}{2}||\beta||_2^2 + \alpha||\beta||_1)), \qquad (1)$$

where λ is shrinkage parameter to be determined by cross validation and $\alpha \in [0, 1]$. The elastic net method includes ridge and LASSO as special cases when taking α to be 0 and 1 respectively.
In this example, we let α increasing from 0 to 1 with step-width 0.001 and perform 10-fold cross validation to decide the optimal shrinkage parameter λ. Using R-package "glment", the smallest validation error is obtained when choosing $\alpha = 0.997$. The test error in that case is 3.26835 reported by Kaggle system. As estimated α is very close to 1, we conclude that the number of actual variables affecting the IC50s is much smaller than 60 and performing variable selection is a necessary step before building prediction models.

2.2 Gradient Boosted Regression Tree

As one simple prediction model may not perform well on the test set, boosting is a technique frequently used to ensemble a group of weak models, typically decision trees so that the performance of the final model is improved. In regression problems, starting with a weak predictor F_m, the algorithm strengthens the predictor by adding an estimator h to get F_{m+1}. At each iteration, F_{m+1} learns to correct its predecessor F_m by fitting h to the residual $y - F_m(x)$.
In gradient tree boosting, at each iteration a regression tree is fitted to the pseudo-residuals and then added to the previous model. The hyper-parameters in gradient tree boosting are number of trees (number of iterations), number of terminal nodes in the trees, maximum tree depth. We choose the set of hyper-parameters that can minimize the validation error.
Using R-package "h2o", the smallest validation error is obtained when choosing number of trees to be 50, maximum tree depth to be 5 and number of terminal nodes in the tree to be 10. The test error in this setting is 3.21231 reported by Kaggle system.

2.3 Local Tree Method

Individual decision trees tend to overfit. Bootstrap-aggregated decision trees combine the results of many decision trees, which reduces the effects of overfitting and improves generalization.Due to large noise in the underlying data set, in the implementation of this tree method, due to some cooperation between gene behaviour, here we raise the 60 features to be 3660 features added with quadratic features. We use the local neighbors of a single point to build a decision tree, which grows in the ensemble using a bootstrap samples of the data and selects a random subset of predictors to use at each decision split as in the random forest algorithm and finally use the model to predict this single point. The least error is 3.18 reported by Kaggle.

2.4 Random Forest

Compared with boosting, random forest achieves prediction error reduction by averaging many trees to reduce variance. In random forest, each tree in the ensemble is built from a sample drawn with replacement, usually bootstrap, from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

Why choose Random Forest

- Mutation status is binary, trees are easy to build;
- Randomly selecting subset of the features, which is suitable for, to some degree, this high dimensional data;

- Underlying relationship between IC50s and 60 mutation status is not necessary linear;
- Generally easy to tuning parameters.

Weakness

- Interpretability is an issue, not many insights for biological guys;
- Data is too sparse which makes trees in the forest too similar, which may interpret why it does not outperform other method;
- Training set is not large, start with high bias/low variance is a better with respect to test error; In addition, how to work its randomization concept well becomes a problem.

Key parameters and results
max_features(The number of features to consider when looking for the best split) $\approx \sqrt{60}$;
max_depth(The maximum depth of the tree) = 7 ($2^7 = 128$ Sample number is 542);
n_estimators(The number of trees in the forest) = 100;
The test error in Kaggle is 3.37.

3 Remark

Group members' contribution:
Problem formulation: Yue Jiang, Zhenzhen Li, Lizhang Miao
Elastic net part: Yue Jiang
Gradient boosted regression tree: Yue Jiang
Local Tree regression: Zhenzhen Li
Random Forest: Lizhang Miao
Report writing: Yue Jiang, Zhenzhen Li, Lizhang Miao

4 Appendix

4.1 R code of elastic net part

```
library(glmnet)
# 10-fold cross validation to decide the optimal lambda
fit.lasso.cv <- cv.glmnet(as.matrix(train[,-1:-3]), train[,3],
    type.measure="mse", alpha=1, family="gaussian")
fit.ridge.cv <- cv.glmnet(as.matrix(train[,-1:-3]), train[,3],
    type.measure="mse", alpha=0, family="gaussian")

mse=c()

for (a in seq(0,1,by=0.00005)) {
  fit.temp <- cv.glmnet(as.matrix(train[,-1:-3]), train[,3],
      type.measure="mse", alpha=a,family="gaussian")
  yhat.temp <-  predict(fit.temp, s=fit.temp$lambda.1se,
      newx=as.matrix(validation[,-1:-3]))
  mse.temp <- mean((validation$IC50s - yhat.temp)^2)
  mse=c(mse,mse.temp)
}

fit.final <- cv.glmnet(as.matrix(train[,-1:-3]), train[,3],
    type.measure="mse", alpha=0.997,family="gaussian")
yhat.final <-  predict(fit.final, s=fit.final$lambda.1se,
    newx=as.matrix(test[,-1:-3]))

sub$IC50<-yhat.final
```

4.2 R code of gradient boosting

```
library(h2o)
h2o.init(nthreads = -1)
h2ovalid<-as.h2o(validation[,-1:-3])
h2otest<-as.h2o(test[,-1:-3])

drug.hex <- h2o.uploadFile(path = "train_drug_500.txt",sep=' ')
independent <- colnames(train)[-1:-3]
dependent <- "IC50s"
gbm_fit<-h2o.gbm(y = dependent, x = independent,
    training_frame = drug.hex, ntrees = 50, max_depth = 5,
    min_rows = 10, stopping_rounds = 4, learn_rate_annealing = 0.9)

#validation set performance
predict.valid<-h2o.predict(gbm_fit, h2ovalid)
predict.valid<-c(as.matrix(predict.valid))
sse.valid<-sum((predict.valid-validation$IC50s)^2)/42

#prediction on test set
predict.test<-h2o.predict(gbm_fit,h2otest)
predict.test<-c(as.matrix(predict.test))

sub$IC50<-predict.test
```

4.3 Matlab Code for Local Tree Regression

```
clear
clc
%% load data
load('data2.mat');
%% set parameters
dist0=16;
%or k_n=16 also works well
bagnum=40;
sigma=1;
XData=[X_train;X_test];
```

```
XData_sub=[XData;X_submission];
Xextend=XData;
Xextend_sub=XData_sub;
for i=1:size(XData,2)
    for j=1:size(XData,2)
        Xextend(:,size(XData,2)+(i-1)*size(XData,2)+j)=XData(:,i).*XData(:,j);
    end
end
for i=1:size(XData_sub,2)
    for j=1:size(XData_sub,2)
        Xextend_sub(:,size(XData,2)+(i-1)*size(XData,2)+j)=XData_sub(:,i).*XDa
    end
end
yData=[y_train;y_test];
%% Calculate distance matrix
%distance 542*542
%distacne_sub 100*542
distance_extend=[];
for i=1:size(yData,1)
    for j=1:size(yData,1)
        distance_extend(i,j)=(Xextend(i,:)-Xextend(j,:))*(Xextend(i,:)-Xextend(
    end
end
distance_sub_extend=[];
for i=1:size(X_submission,1)
    for j=1:size(yData,1)
        distance_sub_extend(i,j)=(Xextend_sub(length(yData)+i,:)-Xextend_sub(j,
    end
end
distance=[];
for i=1:size(yData,1)
    for j=1:size(yData,1)
        distance(i,j)=(XData(i,:)-XData(j,:))*(XData(i,:)-XData(j,:))';
    end
end
distance_sub=[];
for i=1:size(X_submission,1)
    for j=1:size(yData,1)
        distance_sub(i,j)=(XData_sub(length(yData)+i,:)-XData_sub(j,:))*(XData_
    end
end

dist0=18;
y_presub=[];
for i=1:size(X_submission,1)
    i
    neighbor=[];
%     for j=1:length(yData)chosen from 500 data
    for j=1:length(y_train)
        if  distance_sub(i,j)<dist0
```

```
                neighbor=[neighbor j];
            end
        end
        if isempty(neighbor)==1
            y_presub(i)=mean(y_train)
        end
        if isempty(neighbor)==0
        %y_pretest(i-length(y_train))=mean(y_train(neighbor));
        P=[];
        yp=[];
        for pp=1:length(neighbor)
            P(pp,:)=Xextend_sub(neighbor(pp),:);
            yp(pp,1)=yData(neighbor(pp));
        end
             mdl = TreeBagger(bagnum,P,yp,'Method','regression','OOBPrediction','on
             y_presub(i)=predict(mdl,Xextend_sub(i+length(yData),:));
      % end
        end
end
```

4.4 Random Forest

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
train_500 = pd.read_table('train_drug_500.txt', sep = ' ')
test_42 = pd.read_table('test_drug_42.txt', sep = ' ')
pred_set = pd.read_table('submission_set.txt',sep = ' ')
train_500.append(train_500[Y_train.values<0]).append(train_500[Y_train.values<
X_train = train_500.iloc[:,3:]
Y_train = train_500.iloc[:,2]
X_test = test_42.iloc[:, 3:]
Y_test = test_42.iloc[:,2]
rfr = RandomForestRegressor(max_depth=7, n_estimators=100, min_samples_leaf=4,
                            max_features='sqrt', oob_score=True,
                            random_state=333)
rfr.fit(X_train, Y_train)

Y_pred = rfr.predict(X_test)
test_err = sum((Y_pred-Y_test)**2)/42
features_sort = sorted(zip(map(lambda x: round(x, 8), rfr.feature_importances_)
                reverse=True)
print(features_sort)
total_train = train_500.append(test_42)
XX = total_train.iloc[:,3:]
YY = total_train.iloc[:,2]
rfr = RandomForestRegressor(max_depth=7, n_estimators=100, min_samples_leaf=4,
                            max_features='sqrt', oob_score=True,
                            random_state=333)
```

```
rfr.fit(XX, YY)
X_pred = pred_set.iloc[:,3:]
YY_pred = rfr.predict(X_pred)
print(YY_pred)
pp = pd.read_table('OneDrug_submission.csv', sep=',')
pp.iloc[0:,1] = YY_pred
print(pp.iloc[0:,0:])
prediction = pp.to_csv('sub1.csv', index = False)
```