Drug Sensitivity Estimation Based on Empirical Conditional Expectation

ZHAO Yuqi
Department of Mathematics
HKUST

FAN Min
Department of Mathematics
HKUST

PAN Hui
Department of ECE
HKUST

Abstract

The project aims to predict the drug sensitivity of each cell lines given their 60 features, gene mutation status. In this report, we use empirical condition expectation based on the T-test selected features to predict the drug sensitivity. The mean square error of our method gives 3.08. Besides, our methods is interpretive and robust $w.r.t$ hyper-parameter γ.

1 Problem Statement and Data Setup

The dataset consists of 642 cancer cell line samples in response to one drug, Afatinib, with 60 features as gene mutation status. The response y as drug sensitivity is measured by logarithmic IC50, and the feature X_{ij} as mutation status is binary indicating whether the corresponding gene j of cell line i is mutated or not.

A random subset of 100 cell line responses are withdrawn to the test sample. The basic problem is to predict the responses of these test samples based on their binary predictors/features.

Therefore the feature matrix X is n by p and the response vector y is n dimensional, where $n = 542$ since 100 cell line being test sample and $p = 60$. And we want regress y using X under criterion mean square error.

However, most traditional methods have bad performance. It is hard to explain a continuous output variable based on binary input features. Also the training set only contains 642 samples, which is quite limited comparing with the input dimension. This tend to lead to serious overfitting problems. Family of liner regression methods(Lasso, Ridge regression and SVR with polynomial kernel), nearest neighbors methods(SVR with Gaussian kernel) and treelike methods(Gradient Boosting, Random Forest, Decision Tree) gives the mean square error around 3.5 even though the total variance of test set is only 3.22, which state that those methods even can not beat sample mean.

This phenomenon may be given rise by 2 reasons:

1. incompleteness of features
2. drug mechanism that can not be captured by traditional model

As we know, the medicine process might be very complex that related to gene, protein, their geometrical structure and other elements. The gene status may only contain a fraction of information of drug sensitivity (Noticed that in the dataset, there are 32 cell lines whose gene features are all zeros but the variance of the regard responses is up to 2.23). Besides, the mechanism may not be captured by traditional methods because of unknown structure like pathway of gene. All these make the prediction task very challengeable.

2 Methodology

The original regression problem is actually compute conditional expectation $\mathbb{E}(y|X)$ as prediction, where X is the features of a specific sample. Usually, putting some model structure (linear model, nearest neighbors) will give good performance. However in the case that those models do not work well like this project, the reasonable way is to compute the empirical conditional expectation $\hat{\mathbb{E}}(y|X)$. This method is benefit from the binary feature of gene status that we can match the train data to the test one. In this way we can avoid to model the mechanism of drug process.

However, X contain too many dimensions thus lead to small sample size for each configuration and make the estimator unstable. Under this situation, dimension reduction is necessary to make sure sufficient sample size. It must be mentioned that this dimension reduction is actually 'bias-variance trade-off'. The less features, the more bias, but with more samples the variance deduced in some extend.

Based on this methodology, our algorithm consists of two parts, feature selection via p-value and conditional expectation based on selected feature.

2.1 Feature Selection via P-value

In this dataset, the feature space is 60-dimensional. But the specific drug, Afatinib, may only take effect to some few genes. we can use T-test to select the target genes. In particular, to feature i, we can test the hypothesis

$$H_0 : \mathbb{E}(y|X_i = 0) = \mathbb{E}(y|X_i = 1) \quad versus \quad H_1 : \mathbb{E}(y|X_i = 0) \neq \mathbb{E}(y|X_i = 1).$$

The less p-value is, the more significant the drug is to gene mutation feature i. Thus we set the threshold γ to choose the features whose p-value is smaller than γ.

Gene	BRAF	EGFR	ERBB2	NRAS	SMAD4	TP53	EWS_FLI1
p-value	0.0057	0.0059	0.0001	0.0003	0.0117	0.006	0.0013

Table 2.1: Some significant genes and their p-value

2.2 Empirical Conditional Expectation with Adaptive Feature Selection

Once having selected features $\tilde{X}$, we can compute the empirical conditional expectation $\hat{\mathbb{E}}(y|\tilde{X})$ for given cell line features x by simply taking the mean of all samples whose selected features are equal to the given features, in practice

$$\hat{y} = \hat{\mathbb{E}}(y|\tilde{X}) = \sum_{i=1}^{n} y_i \prod_{j \in \tilde{I}} \chi_{\{X_{ij}=x_j\}},$$

where $\tilde{I}$ is the index of selected features and χ characteristic function. Thus

$$C = \sum_{i=1}^{n} \prod_{j \in \tilde{I}} \chi_{\{X_{ij}=x_j\}}$$

is the cardinality of sample set whose feature match the given one cell line. Finally, the $\hat{y}$ would be our prediction for given feature X.

3 Experiment

In the particular experiment, we set our p-value threshold γ as 0.025. Actually, it happened that our results were very robust related to the setting γ. We use 'leave one out' cross validation strategy to test our framework, and the mean square error was around 3.01. which is much better than those of most traditional methods(all around 3.5).

Sending prediction based on our method for test sample to Kaggle in class, the result shows the mean square error is 3.08, which is a relative good answer to this question.

Algorithm 1 Empirical Conditional Expectation

```
Input:
    Training data with label $\{X^{train}, y^{train}\}$ Test data $X^{test}$
Output:
    Predicted value for test $\hat{y}^{test}$
1: Run T-test for each feature in train data, sort those features by p-value
2: Select the features whose corresponding p-value is less then threshold $\gamma$. Let the index of
   selected features be $\tilde{I}$.
3: Match the sample set in train data which have the same configuration with $X^{test}$ in selected
   feature position for $\tilde{I}$, i.e.
       $S = \{(X_i^{train}, y_i^{train}) | \prod_{j \in \tilde{I}} \chi_{\{X_{ij}^{train} = X_j^{test}\}}\} \quad or$
       $S = \{(X_i^{train}, y_i^{train}) | X_{i,\tilde{I}}^{train} = X_{\tilde{I}_k}^{test}\}$
4: Compute the empirical conditional mean
       $\hat{y} = \frac{1}{|S|} \sum_{(X_i, y_i) \in S} y_i$
5: return $\hat{y}$ .
```

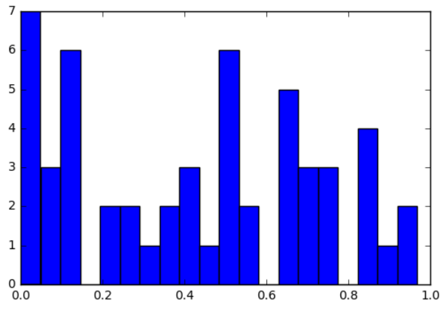

Figure 3.1: Histogram of p-value. x-axis: p-value, y-axis: num of features

4 Conclusion

In this report, we propose a framework based on empirical conditional expectation. This methods is inspired by the feature structure which is in fact binary. Accordingly, we can use training set to match the result of test data and avoid modeling the mechanism of drug process. Notice this framework is highly interpretive, almost tuning free (the result is robust to hyper-parameter γ), and performs well on the given dataset.

Furthermore, this framework is potential with respect to the feature selection process. Notice that we use unified p-value threshold γ which may lead to the imbalance of matching sample set corresponding test data with different configurations. Besides, the interaction between different gene mutations can also be considered in our framework.

5 Appendix: Python Code

Listing 1: main.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
import numpy as np
import pandas as pd

#   Submission
submission = pd.read_csv('OneDrug_submission.csv')

#   Data Setting
df = pd.read_csv('OneDrug_train.csv')

train_index = df.index[np.logical_not(np.isnan(df['IC50s']))]
test_index = df.index[np.isnan(df['IC50s'])]

test = df.values[test_index,3:] # test_feature

feature = df.values[train_index,3:] # train_feature
IC50s = df.values[train_index,2]    # train_y

# feature selection
from scipy.stats import ttest_ind

ttest = [None] * feature.shape[1]
for i in range(feature.shape[1]):
    sample1 = IC50s[ feature[:,i] == 1 ]
    sample0 = IC50s[ feature[:,i] == 0 ]
    if len(sample1) > 1:
        ttest[i] = ttest_ind(sample1, sample0, equal_var = False)
    else:
        ttest[i] = [np.nan]*2
ttest = np.array(ttest)

# new ttest
from learning import adaptive_ttest
model = adaptive_ttest(limit = 15)
model.fit(feature, IC50s, ttest[:,1])
pred = model.predict(test)

submission['IC50'] = pred
submission.to_csv('adaptive_notable0.1.csv',index=False)
```

Listing 2: learnnig.py

```
import numpy as np

class adaptive_ttest(object):
    def __init__(self, limit = 10):
        self.limit = limit

    def fit(self, X, y, p):
        self.X = X
        self.y = y
        self.p = p

    def group_data(X, y):
        idx = list(range(X.shape[0]))
        group = []
        while len(idx)>0:
            standard = X[idx[0],:]
            group.append(np.array([idx[0],y[idx[0]]])[np.newaxis,:])
            del idx[0]
            if len(idx)>0:
                panel = idx.copy()
                for element in panel:
                    if sum(abs(X[element,:] - standard))==0:
                        group[-1] = np.r_[group[-1],
                              np.array([element,y[element]])[np.newaxis,:]]
                        idx.remove(element)
        num_group = len(group)
        number_element = [None]*num_group
        means = [None]*num_group
        variances = [None]*num_group
        data_pos = [None]*num_group
        for i in range(num_group):
            element = group[i]
            number_element[i] = len(element)
            means[i] = element[:,1].mean()
            variances[i] = element[:,1].var()
            data_pos[i] = int(element[0,0])
        number_element = np.array(number_element)[:,np.newaxis]
        means = np.array(means)[:,np.newaxis]
        variances = np.array(variances)[:,np.newaxis]
        data = X[data_pos,:]
        group = np.c_[number_element, means,variances, data]
        return group

    def check(sample, group):
        num_group = len(group)
        data = group[:,3:]
        num_element = group[:,0]
        means = group[:,1]
        variance = group[:,2]
        for i in range(num_group):
            if sum(abs(sample - data[i])) == 0:
                return num_element[i], means[i], variance[i]
        return 0, np.nan, np.nan

    def predict(self, test):
        p_thresholding = np.sort(np.unique(self.p[~np.isnan(self.p)]))
        pred = []
        for sample in test:
            for i in range(len(p_thresholding)):
                print('sample %d/%d, thresholding %d/%d.' %
                      (len(pred), len(test)-1, i, len(p_thresholding)-1))
                p_loc = self.p <= p_thresholding[i]
                group = adaptive_ttest.group_data(self.X[:,p_loc], self.y)
                temp_sample = sample[p_loc]
                num_same, value, var = adaptive_ttest.check(temp_sample, group)
                if num_same <= self.limit:
                    pred.append(value)
                    break
                if i == len(p_thresholding)-1:
                    pred.append(value)
        return pred
```

References

[1] James C Costello. etc. (2014) A community effort to assess and improve drug sensitivity prediction algorithms.Nature Biotechnology 32, 12021212.