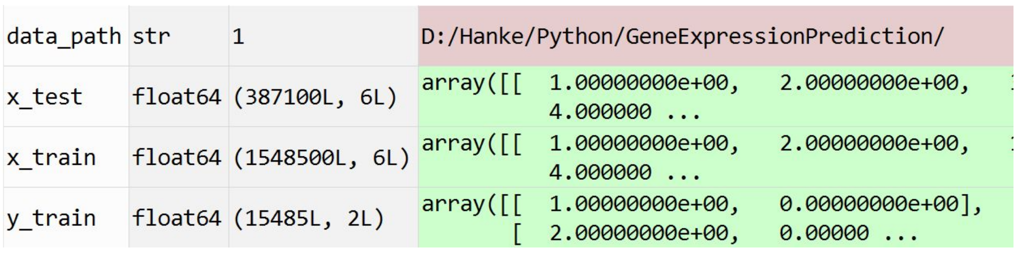
Assignment Report, Group 6

Jingyi Bai 267936
Xiaodong Ming 270441
Wenzhu Xing 267642
Shaptarshi Basu 267647

Task 1: Load data
We loaded data by given template.

```
import numpy as np
import csv
import os

if __name__== '__main__':
    data_path = "D:/Hanke/Python/GeneExpressionPrediction/"

    print("Loading data...")
    x_train = np.loadtxt(data_path + os.sep + "x_train.csv",
                    delimiter = ",", skiprows = 1)
    x_test  = np.loadtxt(data_path + os.sep + "x_test.csv",
                    delimiter = ",", skiprows = 1)
    y_train = np.loadtxt(data_path + os.sep + "y_train.csv",
                    delimiter = ",", skiprows = 1)

    print "All files loaded. Preprocessing..."
```

After loading:



data_path	str	1	D:/Hanke/Python/GeneExpressionPrediction/
x_test	float64	(387100L, 6L)	array([[1.00000000e+00, 2.00000000e+00, 4.000000 ...
x_train	float64	(1548500L, 6L)	array([[1.00000000e+00, 2.00000000e+00, 4.000000 ...
y_train	float64	(15485L, 2L)	array([[1.00000000e+00, 0.00000000e+00], [2.00000000e+00, 0.00000 ...

Task 2: Add a classifier to the template script
We tried Logistic Regression as classifier first, and preprocessed the data by getting its mean and standard value. Accuracy score of Kaggle is 0.88.

```
import csv
from sklearn.linear_model import LogisticRegression
import numpy as np

gene_x_train = np.loadtxt('x_train.csv', skiprows=1, dtype=int, delimiter=',')
gene_x_train_data_list = []
gene_x_train_data_list_mean = []
# gene_x_train_data_list_std = []

gene_y_train = np.loadtxt('y_train.csv', skiprows=1, dtype=int, delimiter=',')
gene_y_train_data_list = []
```

```
gene_x_test = np.loadtxt('x_test.csv', skiprows=1, dtype=int, delimiter=',')
gene_x_test_data_list = []
gene_x_test_data_list_mean = []
# gene_x_test_data_list_std = []

for i in range(len(gene_x_train)):
    if (i % 100) == 0:
        gene_x_train_data_list.append(gene_x_train[i:(i + 100), 1:])
        gene_x_train_data_list_mean.append(np.mean(gene_x_train[i:(i + 100), 1:], axis=0))
        # gene_x_train_data_list_std.append(np.std(gene_x_train[i:(i + 100), 1:], axis=0))

gene_y_train_data_list = gene_y_train[:, 1]

for i in range(len(gene_x_test)):
    if (i % 100) == 0:
        gene_x_test_data_list.append(gene_x_test[i:(i + 100), 1:])
        gene_x_test_data_list_mean.append(np.mean(gene_x_test[i:(i + 100), 1:], axis=0))
        # gene_x_test_data_list_std.append(np.mean(gene_x_test[i:(i + 100), 1:], axis=0))

clf = LogisticRegression()
clf.fit(gene_x_train_data_list_mean, gene_y_train_data_list)
gene_y_test_data_list = clf.predict(gene_x_test_data_list_mean)
gene_y_test_data_list_proba = clf.predict_proba(gene_x_test_data_list_mean)
```

Task 3: Experiment with parameter C and penalty of the LogReg classifier
We used a for loop to find a best C value (bestC) and the corresponding AUC score is maxscore. We choose the C values from the range (10^(-6) , 10^(0)). And the following table is the results of L1 and L2 penalty:

C	L1 penalty	L2 penalty
0.000001	0.499064	0.841392
0.000002	0.499064	0.845201
0.000004	0.499064	0.846750
0.000009	0.499064	0.849334
0.000018	0.556936	0.850561
0.000038	0.530327	0.851272
0.000078	0.519735	0.851142
0.000162	0.798775	0.850239
0.000336	0.835513	0.849076
0.000695	0.840873	0.848688

0.001438	0.846298	0.847783
0.002976	0.848752	0.846298
0.006158	0.851529	0.845910
0.012743	0.851076	0.844941
0.026367	0.850689	0.844619
0.054556	0.847719	0.844038
0.112884	0.846169	0.844103
0.233572	0.845265	0.843909
0.483293	0.845007	0.843650
1.000000	0.844296	0.843586

From this table the best C is 0.00615848211066 with accuracy: 0.85152863135. And the classifier should be: clf = LogisticRegression(penalty='l1'). However, the Accuracy score of Kaggle is still around 0.88.

```
#The definition of classifiers:
clf = LogisticRegression(penalty='l1')
clf = LogisticRegression(penalty='l2')

#C value's range:
C_range = np.logspace(-6, 0, 20)

#The for loop and print the max score and best C value:
for C in C_range:
        clf.C = C
        score = cross_val_score(clf, x_train, y_train, cv=10).mean()
        accuracies.append(score)
        print("C is %f and accuracy is %f" % (C, score))
        if(maxscore <= score):
            maxscore = score
            bestC = C
print "maxscore", maxscore
print "bestC",bestC

#Using the best C to train the model:
clf.C = C
clf.fit(gene_x_train_data_list_mean, gene_y_train_data_list)
gene_y_test_data_list = clf.predict(gene_x_test_data_list_mean)
gene_y_test_data_list_proba = clf.predict_proba(gene_x_test_data_list_mean)
```

Task 4: Add preprocessing and cross-validate

For preprocessing task, firstly we tried to preprocess data by ourselves. What we did is to get the mean value of each column of each gene by np.mean(). Each gene has 100 data.

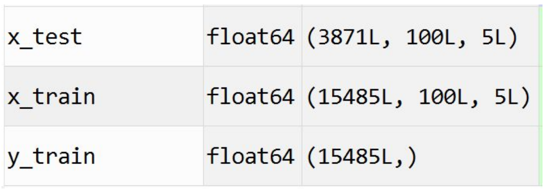
```
import csv
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
import numpy as np

gene_x_train = np.loadtxt('x_train.csv', skiprows=1, dtype=int, delimiter=',')
gene_x_train_data_list = []
gene_x_train_data_list_mean = []
# gene_x_train_data_list_std = []

gene_y_train = np.loadtxt('y_train.csv', skiprows=1, dtype=int, delimiter=',')
gene_y_train_data_list = []

gene_x_test = np.loadtxt('x_test.csv', skiprows=1, dtype=int, delimiter=',')
gene_x_test_data_list = []
gene_x_test_data_list_mean = []
# gene_x_test_data_list_std = []

for i in range(len(gene_x_train)):
    if (i % 100) == 0:
        gene_x_train_data_list.append(gene_x_train[i:(i + 100), 1:])
        gene_x_train_data_list_mean.append(np.mean(gene_x_train[i:(i + 100), 1:], axis=0))
        # gene_x_train_data_list_std.append(np.std(gene_x_train[i:(i + 100), 1:], axis=0))

gene_y_train_data_list = gene_y_train[:, 1]

for i in range(len(gene_x_test)):
    if (i % 100) == 0:
        gene_x_test_data_list.append(gene_x_test[i:(i + 100), 1:])
        gene_x_test_data_list_mean.append(np.mean(gene_x_test[i:(i + 100), 1:], axis=0))
        # gene_x_test_data_list_std.append(np.mean(gene_x_test[i:(i + 100), 1:], axis=0))
```

Then we tried the preprocessing method in given template, but after met with customers and learned basic knowledge about the given data, we thought that keep the data as the given structure will be better. So, we decided to preprocess training data into (15485, 100, 5) shape.

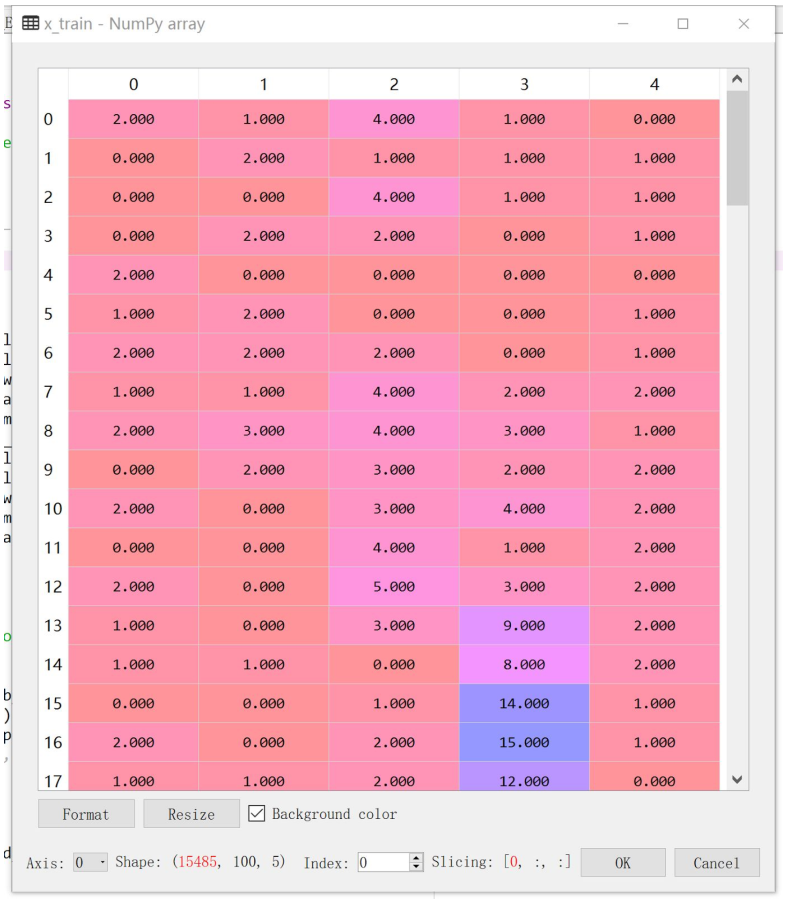
```
    # remove the first column(Id)
    x_train = x_train[:, 1:]
    x_test  = x_test[:, 1:]
    y_train = y_train[:, 1:]

    # Every 100 rows correspond to one gene.
    # Extract all 100-row-blocks into a list using np.split.
    num_genes_train = x_train.shape[0] / 100
    num_genes_test  = x_test.shape[0] / 100

    print("Train / test data has %d / %d genes." %  (num_genes_train, num_genes_test))
    x_train = np.split(x_train, num_genes_train)
```

```
x_test  = np.split(x_test, num_genes_test)

x_train = np.array(x_train)
y_train = np.array(y_train)
x_test  = np.array(x_test)
y_train = np.ravel(y_train)
```

Finally the data structure is as shown in pictures:



x_test	float64	(3871L, 100L, 5L)
x_train	float64	(15485L, 100L, 5L)
y_train	float64	(15485L,)



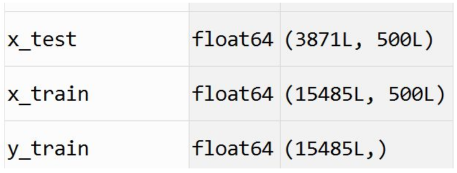
x_train - NumPy array

	0	1	2	3	4
0	2.000	1.000	4.000	1.000	0.000
1	0.000	2.000	1.000	1.000	1.000
2	0.000	0.000	4.000	1.000	1.000
3	0.000	2.000	2.000	0.000	1.000
4	2.000	0.000	0.000	0.000	0.000
5	1.000	2.000	0.000	0.000	1.000
6	2.000	2.000	2.000	0.000	1.000
7	1.000	1.000	4.000	2.000	2.000
8	2.000	3.000	4.000	3.000	1.000
9	0.000	2.000	3.000	2.000	2.000
10	2.000	0.000	3.000	4.000	2.000
11	0.000	0.000	4.000	1.000	2.000
12	2.000	0.000	5.000	3.000	2.000
13	1.000	0.000	3.000	9.000	2.000
14	1.000	1.000	0.000	8.000	2.000
15	0.000	0.000	1.000	14.000	1.000
16	2.000	0.000	2.000	15.000	1.000
17	1.000	1.000	2.000	12.000	0.000

Axis: 0 Shape: (15485, 100, 5) Index: 0 Slicing: [0, :, :]

Task 5: Try different classifiers

In this part we tried different classifiers by data shape which is given in template. Training data has been reshaped by raveling each 100*5 array into a 500-length vector, as follows:



x_test	float64	(3871L, 500L)
x_train	float64	(15485L, 500L)
y_train	float64	(15485L,)

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.lda import LDA
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier,
AdaBoostClassifier, GradientBoostingClassifier

classifiers = [('KNN', KNeighborsClassifier(n_neighbors=10)),
               ('LDA', LDA()),
               # ('SVC', SVC(probability=True)),
               ('LR', LogisticRegression()),
               ('RandomForest', RandomForestClassifier(n_estimators=100)),
               ('ExtraTrees', ExtraTreesClassifier(n_estimators=100)),
               ('AdaBoost', AdaBoostClassifier(n_estimators=100)),
               ('GradientBoosting', GradientBoostingClassifier(n_estimators=100))]
for name, clf in classifiers:
    print(name + ': ')
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    y_pred_proba = clf.predict_proba(x_test)
    print('\n' + name + ' DONE!\n')

    y_test = []
    for i in range(len(y_pred)):
        y_test.append([i+1, y_pred_proba[i][1]])

    filename = name + ".csv"
    print("Writing " + filename + " ...")
    with open(filename, 'w') as f:
        writer = csv.DictWriter(f, fieldnames=["GeneId", "Prediction"], delimiter=",")
        writer.writeheader()
    np.savetxt(filename, y_test, delimiter=",")
    print("Writing DONE.")
    print("-" * 30)

print("\n\n\n------- ALL DONE!!! -------\n\n\n")
```
The best one is Latent Dirichlet Allocation which got 0.89502.

Task 6: Communicate your progress to the customer

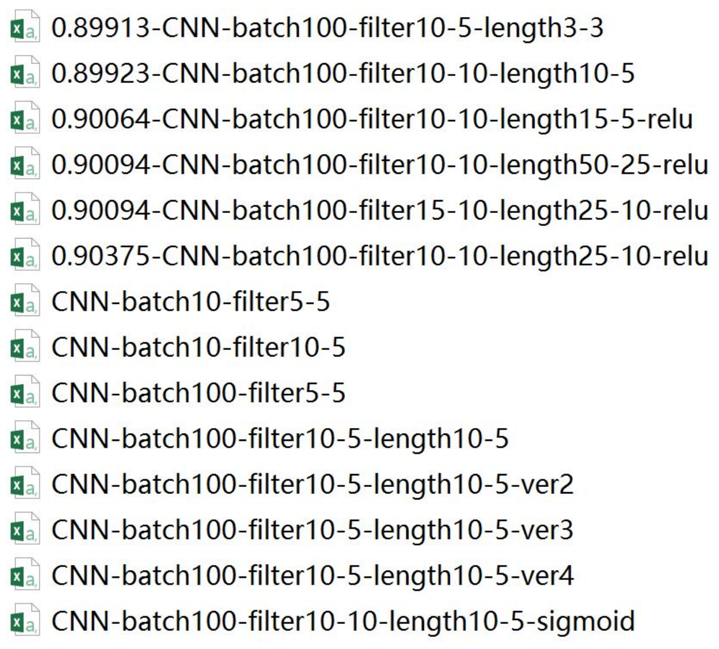
09.02.2017 13:30 -- 15:00	Introduction the background of the topic.
16.02.2017 14:30 -- 15:00	Show our initial methods, results and explain those methods.
23.02.2017 14:30 -- 15:00	Try CNN and explain to the customer.

Task 7: Try deep learning

In this part we used the data in Task 4. The shape of training data is (15485, 100, 5).

```
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.convolutional import Convolution1D
from keras.layers.core import Dense, Activation, Flatten
from sklearn.metrics import accuracy_score, roc_auc_score

    print("CNN...")
    model = Sequential()
    model.add(Convolution1D(nb_filter=100,
                            filter_length=25,
                            activation='relu',
                            input_shape=(x_train.shape[1], x_train.shape[2])))
    model.add(Convolution1D(nb_filter=100,
                            filter_length=10,
                            activation='relu'))
    model.add(Flatten())
    model.add(Dense(1, activation = 'sigmoid'))

    print("Compiling...")
    model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

    print("Fitting...")
    model.fit(x_train, y_train, nb_epoch=10, batch_size=100)
    y_pred = model.predict(x_test)
    y_pred_proba = model.predict_proba(x_test)
    print('\n\nCNN DONE!\n\n')
```

The best accuracy score of Kaggle is 0.90375. All results as follow:



0.89913-CNN-batch100-filter10-5-length3-3
0.89923-CNN-batch100-filter10-10-length10-5
0.90064-CNN-batch100-filter10-10-length15-5-relu
0.90094-CNN-batch100-filter10-10-length50-25-relu
0.90094-CNN-batch100-filter15-10-length25-10-relu
0.90375-CNN-batch100-filter10-10-length25-10-relu
CNN-batch10-filter5-5
CNN-batch10-filter10-5
CNN-batch100-filter5-5
CNN-batch100-filter10-5-length10-5
CNN-batch100-filter10-5-length10-5-ver2
CNN-batch100-filter10-5-length10-5-ver3
CNN-batch100-filter10-5-length10-5-ver4
CNN-batch100-filter10-10-length10-5-sigmoid