ECE9039 – Machine Learning

Chunyang Jia

251001556

1. Filter all “Normal Experiments” by taking into account only active examples “SDS Armed = 1”, and then, merge them in a new file named as “merged exp normal.csv”. Write a script that performs this task and indicate the number of examples of the merged dataset [10 points].

   Answer: first create the empty dataframe. Then loop through the 68 files and filter them by the column Sds_Armed equal to 1. Then append to the Heat normal dataset.

   There are **152356** rows and **10** columns.

```
######Question 1#####################################

import pandas as pd
import numpy as np
import math
path1 = './Datasets/Normal_Experiments/'
path2 = './Datasets/Experiments_with_Anomalies/'

Heat_normal = pd.DataFrame()

#read in the datasets
for number in range(1,69):
    df1 = pd.read_csv(path1+'/HEAT_ID_' +
                      str("{:02d}".format(number)) + '_ALARM_OUT.csv')
    #filter for sds_armed as 1
    df1 = df1[(df1['Sds_Armed'] == 1)]
    Heat_normal = pd.concat([Heat_normal, df1])
# save the normal values file to merged_exp_normal.csv
Heat_normal.to_csv('merged_exp_normal.csv')
```

2. Filter all “Experiments with Anomalies” by taking into account only active examples “SDS Armed = 1” similar to the requirements in Questions 1, and then, merge them in a new file named as “merged exp contains anomalies.csv”. Write a script that performs this task and indicate the number of examples of the merged dataset [10 points].

   Answer: first create the empty dataframe. Then loop through the 19 files and filter them by the column Sds_Armed equal to 1. Then append to the Heat anomalies dataset.

   There are **45626** rows and **10** columns.

```
#####Question 2#####################################

Heat_anomalies = pd.DataFrame()

#read in the datasets
for number in range(1,20):
    df2 = pd.read_csv(path2+'/HEAT_ID_' +
                      str("{:02d}".format(number)) + '_ALARM_OUT_tag.csv')
    #filter for sds_armed as 1
    df2 = df2[(df2['Sds_Armed'] == 1)]
    Heat_anomalies = pd.concat([Heat_anomalies, df2])
# save the normal values file to merged_exp_normal.csv
Heat_anomalies.to_csv('merged_exp_contains_anomalies.csv')
```

3. Since the merged exp contains anomalies.csv contains anomalies, apply any signif- icance test to rank the significance of each feature (X1, X2, ..., X8) as being a distinctive feature of anomalies [5 points].

Answer: I implemented the Chi2 test on the 8 features and printed out the chi2 scores for each feature. X4 and X3 are the two with the highest scores and will be used for the next phase model building.

```
######Question 3#####################################
Heat_normal = pd.read_csv('merged_exp_normal.csv',index_col = 0)
Heat_anomalies = pd.read_csv('merged_exp_contains_anomalies.csv', index_col =0)

X_anomalies = Heat_anomalies.iloc[:,0:8]
y_anomalies = Heat_anomalies.iloc[:,9]

from sklearn.feature_selection import chi2, SelectKBest
chi2_selector = SelectKBest(chi2, k = 8)
chi2_selector.fit(X_anomalies, y_anomalies)

chi2_scores = pd.DataFrame(list(zip(X_anomalies.columns,
              chi2_selector.scores_, chi2_selector.pvalues_)),
              columns=['ftr', 'score', 'pval'])

#rank the features
print (chi2_scores.sort_values(by = 'score', ascending = False))
```

	ftr	score	pval
3	X4	27179.742048	0.000000e+00
2	X3	20765.545612	0.000000e+00
7	X8	19890.345351	0.000000e+00
5	X6	3277.365923	0.000000e+00
4	X5	1023.073345	1.733840e-224
1	X2	583.152894	7.731529e-129
0	X1	568.161416	1.410130e-125
6	X7	287.134248	2.094056e-64

4. Model the normal process “merged exp normal.csv” using Gaussian distribution. Assume that the features are independent. Characterize your model using the following cases:

   – Consider all features (X1, X2, ..., X8) [5 points].
   – Mark the most important two features (obtained from the significance test in Question 3)
   – The projection of the feature space into the first two components using Principle Component Analysis (PCA) (obtained from the significance test in Question 3) [5 points].

```
###4.1 - consider all features
def gaussianMeanVariance(X):
    m,n = X.shape
    sigma = np.zeros(n)
    mean = np.zeros(n)

    mean = np.mean(X, axis = 0)
    sigma = np.var(X, axis = 0)
    return mean,sigma

def computeGaussian(s,mean,sigma):
    #p = 1/(sigma * np.sqrt(2 * math.pi)) * np.exp( - (s - mean)**2 / (2 * sigma**2))
    p = np.exp(-(s-mean)**2/(2*sigma**2))/(math.sqrt(2*math.pi)*sigma)
    return p

def P(X_all_normal):
    mean, sigma = gaussianMeanVariance(X_all_normal)
    p = 1
    for col in range(0, len(X_all_normal.columns)):
        p = p*computeGaussian(X_all_normal.iloc[:,col], mean[col], sigma[col])
    return p

p = P(X_all_normal)
```

```
###4.2 - mark the most important 2 features
def P_toptwo(X_all_normal):
    X_toptwo = X_all_normal.iloc[:,2:4]
    mean, sigma = gaussianMeanVariance(X_toptwo)
    p_toptwo = 1
    for col in range(0, len(X_toptwo.columns)):
        p_toptwo = p_toptwo * computeGaussian(X_toptwo.iloc[:,col], mean[col], sigma[col])
    return p_toptwo

p_toptwo = P_toptwo(X_all_normal)
```

```
###4.3 - PCA principle component analysis
from sklearn.decomposition import PCA

def P_pca(X_all_normal):
    X_pca = X_all_normal
    pca = PCA(n_components = 2)
    X_pca = pca.fit_transform(X_pca)
    X_pca = pd.DataFrame(X_pca)

    mean, sigma = gaussianMeanVariance(X_pca)

    p_pca = 1

    for col in range(0, 2):
        p_pca = p_pca * computeGaussian(X_pca.iloc[:,col], mean[col], sigma[col])
    return p_pca

p_pca = P_pca(X_all_normal)
```

5. Model the same normal process "merged exp normal.csv" using Gaussian distribution with all requirements in Question 4. However, assume that the features are dependent [10 points]. Hint: Think about the co-variance matrix!

```
######Question 5#########################################
###5.1 - consider all features
from scipy.stats import multivariate_normal

def gaussianMeanCovariance(X):

    mean = np.mean(X, axis = 0)
    cov = np.cov(X.T)
    return mean,cov

def P_multigaussian(X_all_normal):
    mean, cov = gaussianMeanCovariance(X_all_normal)
    p_mutilgaussian = multivariate_normal.pdf(X_all_normal, mean = mean , cov = cov)
    return p_mutilgaussian

p_mutilgaussian = P_multigaussian(X_all_normal)
```

```
###5.2 - consider the top 2 features
def P_toptwo_dep(X_all_normal):
    X_toptwo_dep = X_all_normal.iloc[:,2:4]

    mean, cov = gaussianMeanCovariance(X_toptwo_dep)
    p_toptwo_dep = multivariate_normal.pdf(X_toptwo_dep, mean = mean, cov = cov)
    return p_toptwo_dep

p_toptwo_dep = P_toptwo_dep(X_all_normal)
```

```
###5.3 - PCA principle component analysis

def Pca_dep(X_all_normal):

    X_pca_dep = X_all_normal
    pca_dep = PCA(n_components = 2)
    X_pca_dep = pca_dep.fit_transform(X_pca_dep)
    X_pca_dep = pd.DataFrame(X_pca_dep)
    mean, cov = gaussianMeanVariance(X_pca_dep)
    p_pca_dep = multivariate_normal.pdf(X_pca_dep, mean = mean, cov = cov)
    return p_pca_dep

p_pca_dep = Pca_dep(X_all_normal)
```

6. Develop an anomaly alarm by adjusting a threshold ε to your Gaussian models obtained in Questions 3 and 4, and accordingly, generate an alarm accordingly. Use any experiment that contains anomaly as a test case [8 points].

   Six models will be displayed here. I use the mean of the p values as a base then manually tune the models for better results. Further tuning will be in Question 11.

   6.1 Datasets will be splited first and then I will train the model using the functions I defined previously at Question 4.1. Finally, I will print out the accuracy, F1 score, recall and classification report. I manually tuned the threshold, but further tuning can be done in Question 11.

```
######Question 6#############################################
#6 models to be tested, going to take threahod as the mean of all p values
from sklearn.model_selection import train_test_split
from sklearn import metrics
X_anomalies = Heat_anomalies.iloc[:,0:8]
y_anomalies = Heat_anomalies.iloc[:,9]

X_train, X_test,y_train, y_test = train_test_split(X_anomalies, y_anomalies,
                                                   test_size = 0.2,shuffle = False)

X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)

#6.1 - model from 4.1
p_sixone = P(X_train)
threshold_sixone = (np.array(p_sixone)).mean()*0.00001
p_sixone_predict = P(X_test)

y_sixone_predict = []

for val in p_sixone_predict:
    if val <= threshold_sixone:
        y_sixone_predict.append(1)
    elif val > threshold_sixone:
        y_sixone_predict.append(0)

print('6.1 accuracy: ', metrics.accuracy_score(y_sixone_predict,y_test))
print('6.1 F1 score: ',metrics.f1_score(y_sixone_predict,y_test))
print('6.1 recall: ',metrics.recall_score(y_sixone_predict,y_test))
print('6.1 report: ', metrics.classification_report(y_sixone_predict,y_test))
```

```
6.1 accuracy:  0.8915187376725838
6.1 F1 score:  0.6269781461944235
6.1 recall:  0.5323096609085093
6.1 report:                precision    recall  f1-score   support

           0       0.91      0.97      0.94      7563
           1       0.76      0.53      0.63      1563

   micro avg       0.89      0.89      0.89      9126
   macro avg       0.84      0.75      0.78      9126
weighted avg       0.88      0.89      0.88      9126
```

6.2 This model calls for the function I defined previously at Question 4.2. Further tuning will be required.

```
#6.2 - model from 4.2
p_sixtwo = P_toptwo(X_train)
threshold_sixtwo = (np.array(p_sixtwo)).mean()* 13.645
p_sixtwo_predict = P_toptwo(X_test)

y_sixtwo_predict = []

for val in p_sixtwo_predict:
    if val <= threshold_sixtwo:
        y_sixtwo_predict.append(1)
    elif val > threshold_sixtwo:
        y_sixtwo_predict.append(0)

print('6.2 accuracy: ', metrics.accuracy_score(y_sixtwo_predict,y_test))
print('6.2 F1 score: ',metrics.f1_score(y_sixtwo_predict,y_test))
print('6.2 recall: ',metrics.recall_score(y_sixtwo_predict,y_test))
print('6.2 report: ', metrics.classification_report(y_sixtwo_predict,y_test))
```

```
6.2 accuracy:  0.863905325443787
6.2 F1 score:  0.0
6.2 recall:  0.0
6.2 report:                precision    recall  f1-score   support

           0       0.98      0.88      0.93      8975
           1       0.00      0.00      0.00       151

   micro avg       0.86      0.86      0.86      9126
   macro avg       0.49      0.44      0.46      9126
weighted avg       0.96      0.86      0.91      9126
```

6.3 This model calls for the function I defined previously at Question 4.3. Further tuning will be required.

```
#6.3 - model from 4.3
p_sixthree = P_pca(X_train)
threshold_sixthree = (np.array(p_sixthree)).mean() * 1.57555
p_sixthree_predict = P_pca(X_test)

y_sixthree_predict = []

for val in p_sixthree_predict:
    if val <= threshold_sixthree:
        y_sixthree_predict.append(1)
    elif val > threshold_sixthree:
        y_sixthree_predict.append(0)

print('6.3 accuracy: ', metrics.accuracy_score(y_sixthree_predict,y_test))
print('6.3 F1 score: ',metrics.f1_score(y_sixthree_predict,y_test))
print('6.3 recall: ',metrics.recall_score(y_sixthree_predict,y_test))
print('6.3 report: ', metrics.classification_report(y_sixthree_predict,y_test))
```

```
6.3 accuracy:  0.8746438746438746
6.3 F1 score:  0.0
6.3 recall:  0.0
6.3 report:                precision    recall  f1-score   support

           0       0.99      0.88      0.93      9073
           1       0.00      0.00      0.00        53

   micro avg       0.87      0.87      0.87      9126
   macro avg       0.50      0.44      0.47      9126
weighted avg       0.99      0.87      0.93      9126
```

6.4 This model calls for the function I defined previously at Question 5.1. Further tuning will be required.

```
#6.4 - model from 5.1
p_sixfour = P_multigaussian(X_train)
threshold_sixfour = (np.array(p_sixfour)).mean() * 1000
p_sixfour_predict = P_multigaussian(X_test)

y_sixfour_predict = []

for val in p_sixfour_predict:
    if val <= threshold_sixfour:
        y_sixfour_predict.append(1)
    elif val > threshold_sixfour:
        y_sixfour_predict.append(0)

print('6.4 accuracy: ', metrics.accuracy_score(y_sixfour_predict,y_test))
print('6.4 F1 score: ',metrics.f1_score(y_sixfour_predict,y_test))
print('6.4 recall: ',metrics.recall_score(y_sixfour_predict,y_test))
print('6.4 report: ', metrics.classification_report(y_sixfour_predict,y_test))
```

```
6.4 accuracy:  0.8005698005698005
6.4 F1 score:  0.5180084745762713
6.4 recall:  0.3642458100558659
6.4 report:                precision    recall  f1-score   support

           0       0.79      0.98      0.87      6441
           1       0.90      0.36      0.52      2685

   micro avg       0.80      0.80      0.80      9126
   macro avg       0.84      0.67      0.70      9126
weighted avg       0.82      0.80      0.77      9126
```

6.5 This model calls for the function I defined previously at Question 5.2. Further tuning will be required.

```
#6.5 - model from 5.2
p_sixfive = P_toptwo_dep(X_train)
threshold_sixfive = (np.array(p_sixfive)).mean()* 4.15
p_sixfive_predict = P_toptwo_dep(X_test)

y_sixfive_predict = []

for val in p_sixfive_predict:
    if val <= threshold_sixfive:
        y_sixfive_predict.append(1)
    elif val > threshold_sixfive:
        y_sixfive_predict.append(0)

print('6.5 accuracy: ', metrics.accuracy_score(y_sixfive_predict,y_test))
print('6.5 F1 score: ',metrics.f1_score(y_sixfive_predict,y_test))
print('6.5 recall: ',metrics.recall_score(y_sixfive_predict,y_test))
print('6.5 report: ', metrics.classification_report(y_sixfive_predict,y_test))
```

```
6.5 accuracy:  0.5698005698005698
6.5 F1 score:  0.30685028248587576
6.5 recall:  0.19002842772796852
6.5 report:                precision    recall  f1-score   support

           0       0.54      0.95      0.69      4553
           1       0.80      0.19      0.31      4573

   micro avg       0.57      0.57      0.57      9126
   macro avg       0.67      0.57      0.50      9126
weighted avg       0.67      0.57      0.50      9126
```

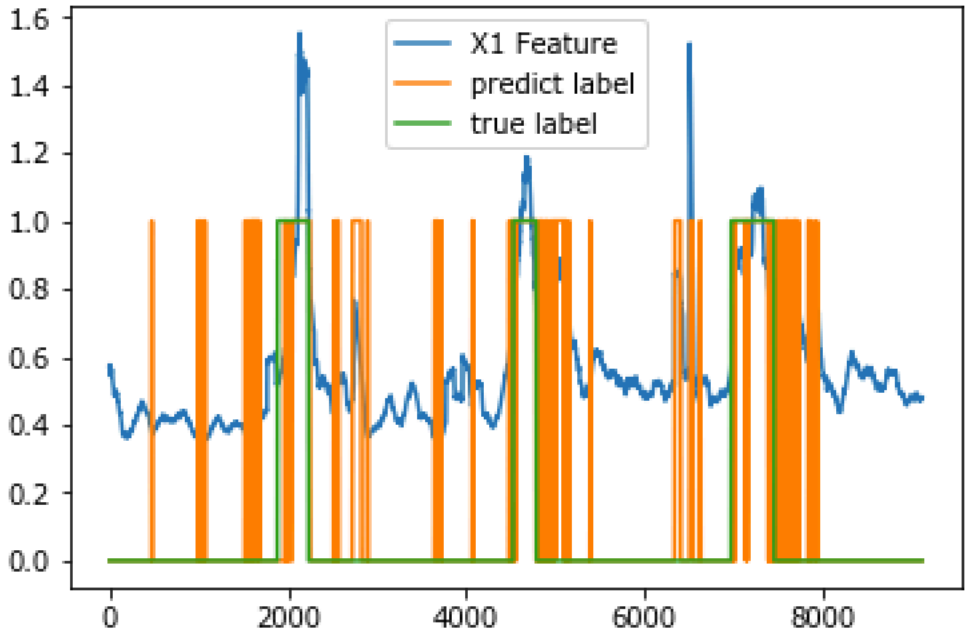
6.6 This model calls for the function I defined previously at Question 5.3. Further tuning will be required.

```
#6.6 - model from 5.3
p_sixsix = Pca_dep(X_train)
threshold_sixsix = p_sixsix.mean()* 1.327
p_sixsix_predict = Pca_dep(X_test)

y_sixsix_predict = []

for val in p_sixsix_predict:
    if val <= threshold_sixsix:
        y_sixsix_predict.append(1)
    elif val > threshold_sixsix:
        y_sixsix_predict.append(0)

print('6.6 accuracy: ', metrics.accuracy_score(y_sixsix_predict,y_test))
print('6.6 F1 score: ',metrics.f1_score(y_sixsix_predict,y_test))
print('6.6 recall: ',metrics.recall_score(y_sixsix_predict,y_test))
print('6.6 report: ', metrics.classification_report(y_sixsix_predict,y_test))
```

```
6.6 accuracy:  0.7019504711812404
6.6 F1 score:  0.15632754342431762
6.6 recall:  0.11814345991561181
6.6 report:                precision    recall  f1-score   support

           0       0.77      0.88      0.82      6993
           1       0.23      0.12      0.16      2133

   micro avg       0.70      0.70      0.70      9126
   macro avg       0.50      0.50      0.49      9126
weighted avg       0.64      0.70      0.66      9126
```

7. Plot the generated alarm, true anomaly flags (given from the dataset), and the feature X1 [5 points].

```
######Question 7#########################################
##will use 6.1 model as an exmaple to show in Question 7
import matplotlib.pyplot as plt
X_x1 = X_test['X1']
predict_label = y_sixone_predict
true_label = y_test

index = list(range(len(X_x1)))

plt.plot(index, X_x1, label = 'X1 Feature')
plt.plot(index, predict_label, label = 'predict label')
plt.plot(index, true_label, label = 'true label')
plt.legend()
plt.show()
```



8. Apply one supervised learning approach for classifying the events to normal and anoma- lies [10 points].

```
#####Question 8##########################################

from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn import metrics

X_all = Heat_anomalies.iloc[:,0:8]
y_all = Heat_anomalies.iloc[:,9]

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_all)
n_x_transform = scaler.transform(X_all)

X_train, X_test, y_train, y_test = train_test_split(n_x_transform,y_all,test_size = 0.3,
                                                    shuffle = False)

clf = SVC(kernel = 'rbf', C = 10, gamma = 'auto')
clf.fit(X_train, y_train)
y_predict_svc = clf.predict(X_test)

print('Accuracy for rbf SVC',metrics.accuracy_score(y_test,y_predict_svc))
print('F1 : ', metrics.f1_score(y_test,y_predict_svc))
print('recall : ', metrics.recall_score(y_test,y_predict_svc))
print('Classification Report: ', metrics.classification_report(y_test,y_predict_svc))
```

```
Accuracy for rbf SVC 0.9061952074810052
F1 :  0.47762408462164363
recall :  0.37293519695044475
Classification Report:                precision    recall  f1-score   support

           0       0.92      0.98      0.95     12114
           1       0.66      0.37      0.48      1574

   micro avg       0.91      0.91      0.91     13688
   macro avg       0.79      0.67      0.71     13688
weighted avg       0.89      0.91      0.89     13688
```

9. Apply any clustering-based algorithm you learn in the class, i.e., (hard and soft clustering with K-means, EM, ..., etc.) to decouple the anomaly data from the normal ones. Is there a direct mapping to the true anomaly tags? discuss your findings [10 points].

```
######Question 9#############################################
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 2, random_state = 0).fit(X_all)
y_predict_kmeans = kmeans.predict(X_all)

print('Accuracy for Kmeans',metrics.accuracy_score(y_all,y_predict_kmeans))
print('F1 for Kmeans',metrics.f1_score(y_all,y_predict_kmeans))
print('recall : ', metrics.recall_score(y_test,y_predict_svc))
print('Classification Report: ', metrics.classification_report(y_all,y_predict_kmeans))
```

```
Accuracy for Kmeans 0.835203611975628
F1 for Kmeans 0.014935150006550504
recall :  0.37293519695044475
Classification Report:                precision    recall  f1-score   support

           0       0.86      0.97      0.91     39195
           1       0.05      0.01      0.01      6431

   micro avg       0.84      0.84      0.84     45626
   macro avg       0.45      0.49      0.46     45626
weighted avg       0.74      0.84      0.78     45626
```

10. Compare the Gaussian-based anomaly detection algorithm, the supervised learning approach you picked, and the clustering approach in terms of [20 points]:

    – Detection capabilities (use the relevant metrics discussed in the class).

    – Time complexity and memory requirements during the training phase.

    – Time complexity and memory requirements during the execution phase.

    Answer:

    1. Detection capabilities. According to the accuracy, F1 score, recall and classification report attached under each model, the Gaussian model at section 6.1 has gotten the best result. 89% accuracy, 63% F1 score and recall of 53%. The gaussian distribution is a good model in this application. However, at this stage, it's hard to tell whether the features are related or not. Further fine tuning is still required, and better results are for sure can be generated. This will be the work to be carried out in Question 11.

       The supervised learning SVC rbf kernel has gotten a 90.6% accuracy, 47% F1 score and 37% recall. Although it has a higher accuracy, but a much lower F1 score and a recall less than 50%. Most of the points are

normal and our goal is to correctly label those anomalies, so in this case, F1 score is a better criterion than accuracy.

K-means has an accuracy of 83.5% but only 1.5% F1 score as it misses most anomaly points. Its performance is very bad.

2. Training Phase: Gaussian-based model's time complexity is O(n), supervised learning is O(n^2), clustering model is O (log n). Memory requirement will be supervised learning requires the most, then Gaussian based models, and lastly clustering model.
3. Execution Phase: Gaussian-based model's time complexity is O(n), supervised learning is O(n^2), clustering model is O (log n). Memory requirement will be supervised learning requires the most, then Gaussian based models, and lastly clustering model.

11. Optimize the parameter ε from Question 6 with the objective of maximizing the detection rate and minimizing the false alarm rate. Compare the results before and after optimizing ε [15 points]. Particularly, consider the following obectives "jointly":

Answer: goal equals to detection rate divided by false alarm rate, so consider F1 score idea is we select a group of thresholds, store the score and find the best one. Use the mean p value as a base then times the coefficient.

The coefficient equals to min max area divided by n equal sections.

```
######Question 11##############################################
"""
goal equals to detection rate divided by false alarm rate, so consider F1 score
idea is we select a group of threhold, store the score and find the best one
After manual tuning, we noticed that the best is among 4 to 5"""

def findbestThreshold(X,y,function,minimum, maximum, n):
    X_train, X_test,y_train, y_test = train_test_split(X, y,
                                                       test_size = 0.2,shuffle = False)
    X_train = pd.DataFrame(X_train)
    X_test = pd.DataFrame(X_test)
    result = pd.DataFrame(columns = ['threhold', 'score'])

    for i in range(0, n):
        p = function(X_train)
        p_predict = function(X_test)
        threshold = (np.array(p)).mean()*(minimum + (maximum - minimum)*i/n)

        y_predict = []
        for val in p_predict:
            if val <= threshold:
                y_predict.append(1)
            elif val > threshold:
                y_predict.append(0)
        score = metrics.f1_score(y_predict, y_test)

        result = result.append({'threhold': threshold, 'score':score},
                               ignore_index = True)

    row = result['score'].argmax()
    best_threhold = result.iat[row, 0]
    best_f1 = result.iat[row, 1]

    return best_threhold,best_f1

T_65,f1_65 = findbestThreshold(X_anomalies, y_anomalies, P_toptwo_dep, 4, 5,5000)
print (T_65,f1_65)
```