```python
#----------------------------------------
#Tool Function
#----------------------------------------
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE, RandomOverSampler
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LogisticRegression, perceptron, SGDClassifier
from sklearn.model_selection import cross_val_score,StratifiedKFold
from sklearn.metrics import accuracy_score,roc_curve,confusion_matrix,precision_recall_curve,auc,f1_score,recall_score,classification_report
from sklearn.metrics import f1_score
from sklearn import svm
from sklearn.utils import shuffle
from sklearn.naive_bayes import GaussianNB,BernoulliNB,ComplementNB
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
def MissingData(data, test_data, method):
    length = data.shape[0]
    if method == 'method1':
        data = shuffle(data,random_state = 17)
        train_label = data['class']
        train_data = data.drop('class',axis=1)
        test_label = test_data['class']
        test_data = test_data.drop('class',axis=1)
        before_drop = train_data.columns.values.tolist()
        train_data = train_data.dropna(thresh=60000*0.3,axis = 1)
        after_drop = train_data.columns.values.tolist()
        diff = set(before_drop).difference(after_drop)
        for name in diff:
            test_data = test_data.drop(name,axis = 1)

        train_data['missing number'] = train_data.isna().sum(axis = 1)
        test_data['missing number'] = test_data.isna().sum(axis = 1)

        for name in train_data.columns.values.tolist():
            if name == 'class':
                pass
            else:
                #if train_data[name].isna().sum(axis = 0)<19999*0.02 and
train_data[name].isna().sum(axis = 0) != 0:
                if train_data[name].isna().sum(axis = 0) != 0:
                    train_data[name] = train_data[name].fillna(train_data[name].median())
                    test_data[name] = test_data[name].fillna(train_data[name].median())
                pass

        train_label = train_label.apply(lambda x: 0 if x=='neg' else 1)
        test_label = test_label.apply(lambda x: 0 if x=='neg' else 1)

    elif method == 'method2':
        train_label = data['class']
        train_data = data.drop('class',axis=1)
        test_label = test_data['class']
        test_data = test_data.drop('class',axis=1)
        before_drop = train_data.columns.values.tolist()
        train_data = train_data.dropna(thresh=60000*0.2,axis = 1)
        after_drop = train_data.columns.values.tolist()
        diff = set(before_drop).difference(after_drop)
        for name in diff:
            test_data = test_data.drop(name,axis = 1)

        train_data['missing number'] = train_data.isna().sum(axis = 1)
        test_data['missing number'] = test_data.isna().sum(axis = 1)

        for name in train_data.columns.values.tolist():
            if name == 'class':
                pass
            else:
                #if train_data[name].isna().sum(axis = 0)<19999*0.02 and
train_data[name].isna().sum(axis = 0) != 0:
                if train_data[name].isna().sum(axis = 0) != 0:
                    train_data[name] = train_data[name].fillna(train_data[name].median())
                    test_data[name] = test_data[name].fillna(train_data[name].median())
                pass

        train_label = train_label.apply(lambda x: 0 if x=='neg' else 1)
        test_label = test_label.apply(lambda x: 0 if x=='neg' else 1)
    print("Missing Data method: .{}".format(method))
```

```python
    return train_data, train_label , test_data, test_label

def Scaler(scaler_method,train_data,test_data):
    if scaler_method == 'standard':
        scaler = StandardScaler()
    else:
        scaler = MinMaxScaler()
    scaler.fit(train_data)
    train_data_scaler = scaler.transform(train_data)
    test_data_scaler = scaler.transform(test_data)
    print("Scaler: .{}".format(scaler_method))
    return train_data_scaler, test_data_scaler

def Feature_selection(feature_choose, train_data_scaler,train_label,test_data_scaler,
param ):
    if feature_choose == 'pca':
        selector = PCA(param[0])
        # selector = PCA(param[0])
        selector.fit(train_data_scaler)
        pass
    elif feature_choose == 'KBest':
        selector = SelectKBest(chi2, param[1])
        selector.fit(train_data_scaler,train_label)
        pass

    print("Feature Selection: .{} ({})".format(feature_choose,param))
    train_data_selection = selector.transform(train_data_scaler)
    test_data_selection = selector.transform(test_data_scaler)
    return train_data_selection, test_data_selection

def Balance(balance_methods, train_data, train_label):
    balance_method = balance_methods[0]
    param = balance_methods[1]
    if balance_method == 'SMOTE':
        if param == 1:
            print('Oversample method: {}'.format('Smote Sampler (1)'))
            balancer = SMOTE(sampling_strategy = 'minority',random_state=17)
            pass
        else:
            print('Oversample method: {}({})'.format('Somte Sampler', param))
            balancer = SMOTE(ratio = param ,random_state=17)
            pass
        pass
    elif balance_method == 'Random':
        print('Oversample method: {}'.format('RandomOverSampler'))
        balancer = RandomOverSampler(sampling_strategy='minority')
        pass
    else:
        print('No Sampler')
        return train_data, train_label
    print("Imbalance Solution: .{}".format(balance_methods))
    train_data_final, train_label_final = balancer.fit_sample(train_data, train_label)
    return train_data_final, train_label_final

def Find_Best_Param(classifier_parameter, train_data_final, train_label_final,foldN):
    clf_kind = classifier_parameter[0]
    parameter = classifier_parameter[1:]
    pN = 19666
    nN = 19666
    cv = StratifiedKFold(n_splits=foldN)
    cost_vec = []
    score_vec = []
    cost_rate_vec = []
    if clf_kind == 'svm':
        c_param = parameter[0]
        gamma_param = parameter[1]
        kernel_param = parameter[2]
        c_vec = []
        gamma_vec = []
        kernel_vec = []
        output_parameter = [clf_kind]
        for c in c_param:
            for gam in gamma_param:
                for kern in kernel_param:
                    c1 = 0
                    c2 = 0
                    cost_final = 0
                    auc_score = 0
                    cost_final = 0
                    print('-------------------------')
                    print("C Parameter :", c)
                    print("Gamma: ", gam)
                    print("kernel: ", kern)
                    print('Training...')
                    for train, val in cv.split(train_data_final, train_label_final):
                        clf = svm.SVC(C=c,gamma=gam,kernel=kern)
                        clf.fit(train_data_final[train],train_label_final[train])
                        y_pred = clf.predict(train_data_final[val])
                        Recall = f1_score(train_label_final[val],y_pred)
                        auc_score += Recall
                        cm = confusion_matrix(train_label_final[val],y_pred).ravel()
                        cost = 500*cm[2]+10*cm[1]
                        c1 += cm[1]
                        c2 += cm[2]
                        cost_final += cost
                    auc_score = auc_score/foldN
                    cost_final=cost_final/foldN
                    cost_rate = 10*c1/nN+500*c2/pN
                    cost_rate_vec.append(cost_rate)
                    cost_vec.append(cost_final)
                    score_vec.append(auc_score)
                    c_vec.append(c)
                    kernel_vec.append(kern)
                    gamma_vec.append(gam)
                    print ('F1 score  =',auc_score)
                    print('Cost = ' , cost_final)
                    print('Cost Rate = ', cost_rate)
                    print('-------------------------')
        ind_max = cost_vec.index(min(cost_vec))
        best_c = c_vec[ind_max]
        best_gamma = gamma_vec[ind_max]
        best_kernel = kernel_vec[ind_max]
        output_parameter.append(best_c)
        output_parameter.append(best_gamma)
        output_parameter.append(best_kernel)
    elif clf_kind == 'logisticRegression':
        c_param = parameter[0]
        penaltys = parameter[1]
        c_vec = []
        penalty_vec = []
        output_parameter = [clf_kind]
        for c in c_param:
            for penal in penaltys:
```

```python
                c1 = 0
                c2 = 0
                auc_score = 0
                print('------------------------')
                print("C Parameter :", c)
                print("Penalty: ", penal)
                print('------------------------')
                cost_final = 0
                for train, val in cv.split(train_data_final, train_label_final):
                    clf = LogisticRegression(C = c, penalty = penal ,solver='liblinear')
                    clf.fit(train_data_final[train],train_label_final[train])
                    y_pred = clf.predict(train_data_final[val])
                    Recall = f1_score(train_label_final[val],y_pred)
                    auc_score += Recall
                    cm = confusion_matrix(train_label_final[val],y_pred).ravel()
                    cost = 500*cm[2]+10*cm[1]
                    c1 += cm[1]
                    c2 += cm[2]
                    cost_final += cost
                cost_final/foldN
                cost_final=cost_final/foldN
                cost_rate = 10*c1/nN+500*c2/pN
                cost_rate_vec.append(cost_rate)
                auc_score = auc_score/foldN
                cost_vec.append(cost_final)
                score_vec.append(auc_score)
                c_vec.append(c)
                penalty_vec.append(penal)
                print ('F1 score =',auc_score)
                print('Cost = ' , cost_final)
                print('Cost Rate = ', cost_rate)
                print('-------------------------')
                print('')
        ind_max = cost_vec.index(min(cost_vec))
        best_c = c_vec[ind_max]
        best_penalty = penalty_vec[ind_max]
        output_parameter.append(best_c)
        output_parameter.append(best_penalty)
    elif clf_kind == 'SGDperceptron':
        penaltys = parameter[0]
        penalty_vec = []
        output_parameter = [clf_kind]
        for penalty_n in penaltys:
            auc_score = 0
            print('------------------------')
            print("Penalty: ", penalty_n)
            cost_final = 0
            c1 = 0
            c2 = 0
            for train, val in cv.split(train_data_final, train_label_final):
                clf = SGDClassifier(loss='perceptron', penalty= penalty_n)
                clf.fit(train_data_final[train],train_label_final[train])
                y_pred = clf.predict(train_data_final[val])
                Recall = f1_score(train_label_final[val],y_pred)
                auc_score += Recall
                cm = confusion_matrix(train_label_final[val],y_pred).ravel()
                cost = 500*cm[2]+10*cm[1]
                c1 += cm[1]
                c2 += cm[2]
                cost_final += cost
            cost_final=cost_final/foldN
            cost_rate = 10*c1/nN+500*c2/pN
            cost_rate_vec.append(cost_rate)
            cost_vec.append(cost_final)
            auc_score = auc_score/foldN
            score_vec.append(auc_score)
            penalty_vec.append(penalty_n)
            print ('F1 score =',auc_score)
            print('Cost = ' , cost_final)
            print('Cost Rate = ', cost_rate)
            print('-------------------------')
            print('')
        ind_max = cost_vec.index(min(cost_vec))
        best_penalty = penalty_vec[ind_max]
        output_parameter.append(best_penalty)

    elif clf_kind == 'NB':
        kinds = parameter[0]
        output_parameter = [clf_kind]
        print('-------------------------')
        for kind in kinds:
            auc_score = 0
            cost_final = 0
            c1 = 0
            c2 = 0
            if kind == 'gaussian':
                clf = GaussianNB()
                print('GaussianNB:')
            elif kind == 'ber':
                clf = BernoulliNB()
                print('BernoulliNB:')
            else:
                clf = ComplementNB()
                print('ComplementNB')
                pass
            for train, val in cv.split(train_data_final, train_label_final):
                clf.fit(train_data_final[train],train_label_final[train])
                y_pred = clf.predict(train_data_final[val])
                Recall = f1_score(train_label_final[val],y_pred)
                auc_score += Recall
                cm = confusion_matrix(train_label_final[val],y_pred).ravel()
                cost = 500*cm[2]+10*cm[1]
                cost_final += cost
                c1 += cm[1]
                c2 += cm[2]
            cost_rate = 10*c1/nN+500*c2/pN
            cost_rate_vec.append(cost_rate)
            cost_final=cost_final/foldN
            cost_vec.append(cost_final)
            auc_score = auc_score/foldN
            score_vec.append(auc_score)
            print ('F1 score  =',auc_score)
            print('Cost = ' , cost_final)
            print('Cost Rate = ', cost_rate)
            print('-------------------------')
        ind_max = cost_vec.index(min(cost_vec))
        best_NB = kind[ind_max]
        output_parameter.append(best_NB)

    elif clf_kind == 'KNN':
        output_parameter=[clf_kind]
        ks = parameter[0]
        k_vec = []
        for k in ks:
            auc_score = 0
            c1 = 0
            c2 = 0
            print('------------------------')
            print("K: ", k)
            print('------------------------')
            cost_final = 0
            for train, val in cv.split(train_data_final, train_label_final):
                clf = KNeighborsClassifier(n_neighbors= k )
                clf.fit(train_data_final[train],train_label_final[train])
                y_pred = clf.predict(train_data_final[val])
                Recall = f1_score(train_label_final[val],y_pred)
                auc_score += Recall
                cm = confusion_matrix(train_label_final[val],y_pred).ravel()
                cost = 500*cm[2]+10*cm[1]
                cost_final += cost
                c1 += cm[1]
                c2 += cm[2]
            cost_rate = 10*c1/nN+500*c2/pN
            cost_rate_vec.append(cost_rate)
            cost_final=cost_final/foldN
            cost_vec.append(cost_final)
            auc_score = auc_score/foldN
            score_vec.append(auc_score)
            k_vec.append(k)
            print ('F1 score =',auc_score)
            print('Cost = ' , cost_final)
            print('Cost Rate = ', cost_rate)
            print('-------------------------')

        ind_max = cost_vec.index(min(cost_vec))
        best_k = k_vec[ind_max]
        output_parameter.append(best_k)
    elif clf_kind == 'NN':
        output_parameter=[clf_kind]
        layers = parameter[0]
        layer_vec = []
        for layer in layers:
            auc_score = 0
            c1 = 0
            print('------------------------')
            print("Layers : ", layer)
            print('------------------------')
            c2 = 0
            cost_final = 0
            for train, val in cv.split(train_data_final, train_label_final):
                clf = MLPClassifier(hidden_layer_sizes= layer)
                clf.fit(train_data_final[train],train_label_final[train])
                y_pred = clf.predict(train_data_final[val])
                Recall = f1_score(train_label_final[val],y_pred)
                auc_score += Recall
                cm = confusion_matrix(train_label_final[val],y_pred).ravel()
                cost = 500*cm[2]+10*cm[1]
                cost_final += cost
                c1 += cm[1]
                c2 += cm[2]
            cost_rate = 10*c1/nN+500*c2/pN
            cost_rate_vec.append(cost_rate)
            cost_final=cost_final/foldN
            cost_vec.append(cost_final)
            auc_score = auc_score/foldN
            score_vec.append(auc_score)
            layer_vec.append(layer)
            print ('F1 score =',auc_score)
            print('Cost = ' , cost_final)
            print('Cost Rate = ', cost_rate)
            print('-------------------------')

        ind_max = cost_vec.index(min(cost_vec))
        best_layer = layer_vec[ind_max]
        output_parameter.append(best_layer)
    else:
        clf = svm.SVC(C=0.01,gamma=0.01,kernel='rbf')
        output_parameter.append(0.01)
        output_parameter.append(0.01)
        output_parameter.append('rbf')
    return output_parameter,cost_vec,cost_rate_vec,score_vec



def Classifier(classifier_parameter, train_data_final, train_label_final,
text_data_final, test_label, loop):
    clf_kind = classifier_parameter[0]
    parameter = classifier_parameter[1:]
    final = 0
    print('Parameter: ',classifier_parameter)
    cm_final = np.zeros(4)
    for i in range(loop):
        if clf_kind == 'svm':
            clf = svm.SVC(C=parameter[0],gamma=parameter[1],kernel=parameter[2])
        elif clf_kind == 'logisticRegression':
            clf = LogisticRegression(C = parameter[0], penalty =
parameter[1] ,solver='liblinear')
        elif clf_kind == 'SGDperceptron':
            clf = SGDClassifier(loss='perceptron', penalty= parameter[0])
        elif clf_kind == 'GaussianNB':
            clf = GaussianNB()
        elif clf_kind == 'KNN':
            clf = KNeighborsClassifier(n_neighbors = parameter[0])
        elif clf_kind == 'NN':
            clf = MLPClassifier(hidden_layer_sizes=parameter[0])
        else:
            clf = svm.SVC(C=0.01,gamma=0.01,kernel='rbf')

        clf.fit(train_data_final, train_label_final)
        y_pred_test = clf.predict(text_data_final)
        recall_test_per=f1_score(test_label,y_pred_test)
        final += recall_test_per
        cm = confusion_matrix(test_label,y_pred_test).ravel()
        cm_final += cm
    final_score = final/loop
    cm_final = cm_final/loop
    print ('Final F1 score for Classifier
(.{})= .{}'.format(classifier_parameter,final_score))
    print('--------------------------')
    print('')
    cm_final = pd.DataFrame(cm_final.reshape((1,4)), columns=['TN', 'FP', 'FN', 'TP'])
    print(cm_final)
    cost = 500*cm_final['FN']+10*cm_final['FP']
    print('The final cost is : {}'.format(cost))
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score,StratifiedKFold
from sklearn.metrics import
accuracy_score,roc_curve,confusion_matrix,precision_recall_curve,auc,roc_auc_sc
ore,recall_score,classification_report
from sklearn.metrics import f1_score
import util as md

missing_data_methods = ['method2']
#preprocess_methods = [['standard','pca'],['minmax','KBest']]
#missing_data_method = ['method2']
preprocess_method = ['standard','pca']
balance_method = ['SMOTE']
# balance_method = ['No',0]

# classifier_parameter = ['svm',[1],[0.01],['rbf']] #classifier 1
#classifier_parameter = ['SGDperceptron',['l1','l2']]
#classifier_parameter =['logisticRegression',[0.1,1,10,100],['l2']]
classifier_parameter1 = ['KNN',[2]]
classifier_parameter2 = ['NN',[50]]
classifier_parameter3 = ['svm',[1],[0.01],['rbf']]
#classifier_parameter =['NB',['gaussian','ber']]
foldN = 5
loop = 1
cost_vector1 = []
cost_rate_vector1 = []
score_vector1 = []
cost_vector2 = []
cost_rate_vector2 = []
score_vector2 = []
cost_vector3 = []
cost_rate_vector3 = []
score_vector3 = []
param1 = [0.95,60]
params =range(19,99,2)
```

```
i = 1
print('Start...')
data =
pd.read_csv("/Users/weizhongjin/usc/ee559/finaldata/aps_failure_training_set_SM
ALLER.csv" , na_values='na')
test =
pd.read_csv("/Users/weizhongjin/usc/ee559/finaldata/aps_failure_test_set.csv" ,
na_values='na')
for missing_data_method in missing_data_methods:
    for param in params:
    #for preprocess_method in preprocess_methods:
        param = param/100
        balance_method_all = [balance_method,param]
        print('-------------------------')
        print('Parameter:')
        scaler_method = preprocess_method[0]
        feature_choose = preprocess_method[1]
        train_data, train_label , test_data, test_label = md.MissingData(data,
test, missing_data_method)
        train_data = np.array(train_data)
        test_data = np.array(test_data)
        train_data, test_data = md.Scaler(scaler_method,train_data,test_data)
        train_data, test_data = md.Feature_selection(feature_choose,
train_data,train_label, test_data,param1)
        train_data, train_label =
md.Balance(balance_method_all,train_data,train_label)
        final_classifier_parameter1,cost_vec1,cost_rate_vec1,score_vec1 =
md.Find_Best_Param(classifier_parameter1, train_data, train_label,foldN)
        final_classifier_parameter2,cost_vec2,cost_rate_vec2,score_vec2 =
md.Find_Best_Param(classifier_parameter2, train_data, train_label,foldN)
        final_classifier_parameter3,cost_vec3,cost_rate_vec3,score_vec3 =
md.Find_Best_Param(classifier_parameter3, train_data, train_label,foldN)
        cost_vector1.append(cost_vec1)
        cost_rate_vector1.append(cost_rate_vec1[0])
        score_vector1.append(score_vec1)
        cost_vector2.append(cost_vec2)
        cost_rate_vector2.append(cost_rate_vec2[0])
        score_vector2.append(score_vec2)
        cost_vector3.append(cost_vec3)
        cost_rate_vector3.append(cost_rate_vec3[0])
        score_vector3.append(score_vec3)
```

```
       # print('Best classifier
parameter :{}'.format(final_classifier_parameter))
       # print('Cost vector: {}'.format(cost_vec))
       # print('Cost Rate vector: {}'.format(cost_rate_vec))
       # print('Score vector: {}'.format(score_vec))
print('/---------------------------------------/')
cost_rate_vectors1 = np.array(cost_vector1)
cost_rate_vectors2 = np.array(cost_vector2)
cost_rate_vectors3 = np.array(cost_vector3)
plt.title('Comparison of different Smot Parameter')
plt.plot(params,cost_rate_vectors1,color='blue', label='KNN')
plt.plot(params,cost_rate_vectors2,color='red', label='SVM')
plt.plot(params,cost_rate_vectors3,color='green', label='MLP(NN)')
plt.legend()
plt.xlabel('Smote Ratio')
plt.ylabel('Cost')
plt.show()
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score,StratifiedKFold
from sklearn.metrics import
accuracy_score,roc_curve,confusion_matrix,precision_recall_curve,auc,roc_auc_sc
ore,recall_score,classification_report
from sklearn.metrics import f1_score
import util as md

missing_data_methods = ['method2']
#preprocess_methods = [['standard','pca'],['minmax','KBest']]
missing_data_method = ['method2']
preprocess_method = ['standard','pca']
balance_method = ['SMOTE',1]
# balance_method = ['No',0]

# classifier_parameter = ['svm',[1],[0.01],['rbf']] #classifier 1
#classifier_parameter = ['SGDperceptron',['l1','l2']]
#classifier_parameter =['logisticRegression',[0.1,1,10,100],['l2']]
classifier_parameter1 = ['KNN',[2]]
classifier_parameter2 = ['NN',[50]]
classifier_parameter3 = ['svm',[1],[0.01],['rbf']]
#classifier_parameter =['NB',['gaussian','ber']]
foldN = 5
loop = 1
cost_vector1 = []
cost_rate_vector1 = []
score_vector1 = []
cost_vector2 = []
cost_rate_vector2 = []
score_vector2 = []
cost_vector3 = []
cost_rate_vector3 = []
score_vector3 = []
#5param = [0.95,60]
params =range(19,99,2)
```

```
i = 1
print('Start...')
data =
pd.read_csv("/Users/weizhongjin/usc/ee559/finaldata/aps_failure_training_set_SM
ALLER.csv" , na_values='na')
test =
pd.read_csv("/Users/weizhongjin/usc/ee559/finaldata/aps_failure_test_set.csv" ,
na_values='na')
for missing_data_method in missing_data_methods:
    for param in params:
    #for preprocess_method in preprocess_methods:
        #balance_method_all = [balance_method,1]
        param = param/100
        param = [param]
        print('[Combination {}]'.format(i))
        i = i+1
        print('-------------------------')
        print('Parameter:')
        scaler_method = preprocess_method[0]
        feature_choose = preprocess_method[1]
        train_data, train_label , test_data, test_label = md.MissingData(data,
test, missing_data_method)
        train_data = np.array(train_data)
        test_data = np.array(test_data)
        train_data, test_data = md.Scaler(scaler_method,train_data,test_data)
        train_data, test_data = md.Feature_selection(feature_choose,
train_data,train_label, test_data,param)
        train_data, train_label =
md.Balance(balance_method,train_data,train_label)
        final_classifier_parameter1,cost_vec1,cost_rate_vec1,score_vec1 =
md.Find_Best_Param(classifier_parameter1, train_data, train_label,foldN)
        final_classifier_parameter2,cost_vec2,cost_rate_vec2,score_vec2 =
md.Find_Best_Param(classifier_parameter2, train_data, train_label,foldN)
        final_classifier_parameter3,cost_vec3,cost_rate_vec3,score_vec3 =
md.Find_Best_Param(classifier_parameter3, train_data, train_label,foldN)
        cost_vector1.append(cost_vec1)
        cost_rate_vector1.append(cost_rate_vec1[0])
        score_vector1.append(score_vec1)
        cost_vector2.append(cost_vec2)
        cost_rate_vector2.append(cost_rate_vec2[0])
        score_vector2.append(score_vec2)
        cost_vector3.append(cost_vec3)
        cost_rate_vector3.append(cost_rate_vec3[0])
        score_vector3.append(score_vec3)
        # print('Best classifier
parameter :{}'.format(final_classifier_parameter))
        # print('Cost vector: {}'.format(cost_vec))
        # print('Cost Rate vector: {}'.format(cost_rate_vec))
        # print('Score vector: {}'.format(score_vec))
print('/---------------------------------------/')
cost_rate_vectors1 = np.array(cost_vector1)
cost_rate_vectors2 = np.array(cost_vector2)
cost_rate_vectors3 = np.array(cost_vector3)
plt.title('Comparison of different PCA Parameter')
plt.plot(params,cost_rate_vectors1,color='blue', label='KNN')
plt.plot(params,cost_rate_vectors2,color='red', label='SVM')
plt.plot(params,cost_rate_vectors3,color='green', label='MLP(NN)')
plt.legend()
plt.xlabel('PCA Parameter')
plt.ylabel('Cost')
plt.show()
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score,StratifiedKFold
from sklearn.metrics import accuracy_score,roc_curve,confusion_matrix,precision_recall_curve,auc,roc_auc_score,recall_score,classification_report
from sklearn.metrics import f1_score
import util as md

missing_data_methods = ['method1','method2']
preprocess_methods = [['standard','pca'],['minmax','KBest']]
#missing_data_method = ['method2']
#preprocess_method = ['standard','pca']
balance_method = ['SMOTE',1]
# balance_method = ['No',0]

# classifier_parameter = ['svm',[0.1,1,10,100],[0.01],['rbf','linear','sigmoid']] #classifier 1
#classifier_parameter = ['SGDperceptron',['l1','l2']]
classifier_parameter =['logisticRegression',[0.1,1,10,100],['l2']]
#classifier_parameter =['NB',['gaussian','ber']]
#classifier_parameter =['NN',[50,100,150]]
foldN = 5
loop = 1
param1 = [0.95,60]
params =range(19,99,2)
i = 1
print('Start...')
data = pd.read_csv("/Users/weizhongjin/usc/ee559/finaldata/aps_failure_training_set_SMALLER.csv" , na_values='na')
test = pd.read_csv("/Users/weizhongjin/usc/ee559/finaldata/aps_failure_test_set.csv" , na_values='na')
for missing_data_method in missing_data_methods:
   for preprocess_method in preprocess_methods:
       print('--------------------------')
       print('Parameter:')
       scaler_method = preprocess_method[0]
       feature_choose = preprocess_method[1]
       train_data, train_label , test_data, test_label = md.MissingData(data,
test, missing_data_method)
       train_data = np.array(train_data)
       test_data = np.array(test_data)
       train_data, test_data = md.Scaler(scaler_method,train_data,test_data)
       train_data, test_data = md.Feature_selection(feature_choose,
train_data,train_label, test_data,param1)
       train_data, train_label =
md.Balance(balance_method,train_data,train_label)
       final_classifier_parameter,cost_vec,cost_rate_vec,score_vec =
md.Find_Best_Param(classifier_parameter, train_data, train_label,foldN)
       print('Best classifier parameter :{}'.format(final_classifier_parameter))
       print('Cost vector: {}'.format(cost_vec))
       print('Cost Rate vector: {}'.format(cost_rate_vec))
       print('Score vector: {}'.format(score_vec))
print('/---------------------------------------/')
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score,StratifiedKFold
from sklearn.metrics import accuracy_score,roc_curve,confusion_matrix,precision_recall_curve,auc,roc_auc_score,recall_score,classification_report
from sklearn.metrics import f1_score
import util as md

missing_data_method = ['method2']
preprocess_method = ['standard','pca']
balance_method = ['SMOTE',1]
classifier_parameter1 = ['KNN',2]
classifier_parameter2 = ['svm',1,0.01,'rbf']
classifier_parameter3 = ['NN',50]
foldN = 5
loop = 1
param = [0.95,60]
print('Start...')
data = pd.read_csv("/Users/weizhongjin/usc/ee559/finaldata/aps_failure_training_set.csv" , na_values='na')
test = pd.read_csv("/Users/weizhongjin/usc/ee559/finaldata/aps_failure_test_set.csv" , na_values='na')
print('-------------------------')
print('Parameter:')
scaler_method = preprocess_method[0]
feature_choose = preprocess_method[1]
train_data, train_label , test_data, test_label = md.MissingData(data, test, missing_data_method[0])
train_data = np.array(train_data)
test_data = np.array(test_data)
train_data, test_data = md.Scaler(scaler_method,train_data,test_data)
train_data, test_data = md.Feature_selection(feature_choose, train_data,train_label, test_data,param)
```

```
train_data, train_label =
md.Balance(balance_method,train_data,train_label)
print('/---------------------------------------/')
final_classifier_parameter = classifier_parameter3
md.Classifier(final_classifier_parameter, train_data,
train_label,test_data,test_label,loop)
```