*#----------------------------------------*

*#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))

**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:

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))

**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 = ['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\_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** 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()

**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 = ['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\_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** 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()

**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**('/---------------------------------------/')

**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)