1. Problem Definition

The project mainly focused on classifying the dataset, which is from The Lung Cancer Database Project at the XXX Cancer Center. It is classified into two groups with the labels of malignant and benign.

30 different properties of lung cells will be used to form a logistic regression model for predicting and indicating the diagnosis whether the cell is malignant or benign. In the following, we will use 1 to represent malignant and 0 to represent benign. The given training dataset will be used to train model and the given testing dataset will be used to predict the unknown labels.

After building the logistic regression model, regularization method will be used to reduce the effect of overfitting when the model has less generalization ability. Since the regularization can adjust the contribution of features to the model when doing the prediction of outcome, the method with using regularization is good for us to keep all features to build the logistic regression model.

The implementation will be using Python to build the model, to fit the data and to evaluate after the model construction with given dataset.

2. Model Design and Considerations

Before the model building and model fitting, we will do scaling with the dataset for features in both training and testing dataset first, in order to prevent the overflow issue of Python system.

Since we want to identify the situation of lung cells, which is about whether the cells are malignant or benign, we will use the information of cells with various samples to model and to figure out the relationship between these information and dependent variables. 30 different features, which are the properties of lung cells, will be used to train a logistic regression model as the dependent variable is binary and designed into two classes. The model building will be using the logistic regression with implying gradient decent algorithm in Python to refine the value of the parameters for minimizing the cost function.

Also, by considering the quantity of features that will be used, the state of overfitting may happen in the analysis. It will be handled by implementing regularization method with the logistic regression algorithm to reduce the error of having high variance. And better values of the parameters will be found, which are the coefficient of the features, after solving the influence for the features by regularization.

We will utilize 5-folds cross validation to implement the given training dataset of both attributes' dataset ("X_train.csv") and label's dataset ("Y_train.csv") into 5 subsets of training and testing dataset. 4 (K minus 1) folds will be used to be the training data for training the model and the remaining data is treated as the testing data every time. The main reason to split the dataset is that it can let us check the accuracy of the constructed model, as the given testing data does not have corresponding true label for comparing. Therefore, it can validate every model after model-fitting by splitting dataset of the completed dataset with both having predicted values and true values in labels. Also, it can choose the model with reducing the effect of overfitting and underfitting by using the one equals the median of the accuracy of the five models.

In the model evaluation part, the model performance can be tested by accuracy, precision, recall and F1-score. Since these testing scores are useful for judging how good of the model of implementing classification algorithm is, we can see the percentage of corrected results or relevant results.

3. Solutions and Implementation Details

First, some packages which are required for later use of different perspective will be imported, such as “sklearn.preprocessing” package for scaling the data and “statistics” module for calculating the median or mean of the data.

```
import pandas as pd
from sklearn import preprocessing
import numpy as np
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
import statistics
```

Second, the given dataset of attributes and labels of corresponding training and testing data as “x_train”, “y_train” and “x_test” will be imported. With the data preprocessing part, the two datasets about the properties of lung cells will have the scaling before using. We can see the difference of before and after scaling in the below figure.

```
x_train = np.loadtxt('X_train.csv', dtype=float, delimiter=',')
x_train_s = preprocessing.scale(x_train)
y_train = np.loadtxt('Y_train.csv', dtype=float, delimiter=',')
x_test = np.loadtxt('X_test.csv', dtype=float, delimiter=',')
x_test_s = preprocessing.scale(x_test)
```

```
#Before scaling
print(x_train[0])
print(y_train[0])
print(x_test[0])
#After scaling
print(x_train_s[0])
print(x_test_s[0])
```

```
[2.018e+01 1.954e+01 1.338e+02 1.250e+03 1.133e-01 1.489e-01 2.133e-01
 1.259e-01 1.724e-01 6.053e-02 4.331e-01 1.001e+00 3.008e+00 5.249e+01
 9.087e-03 2.715e-02 5.546e-02 1.910e-02 2.451e-02 4.005e-03 2.203e+01
 2.507e+01 1.460e+02 1.479e+03 1.665e-01 2.942e-01 5.308e-01 2.173e-01
 3.032e-01 8.075e-02]
1.0
[1.114e+01 1.407e+01 7.124e+01 3.846e+02 7.274e-02 6.064e-02 4.505e-02
 1.471e-02 1.690e-01 6.083e-02 4.222e-01 8.092e-01 3.330e+00 2.884e+01
 5.541e-03 3.387e-02 4.505e-02 1.471e-02 3.102e-02 4.831e-03 1.212e+01
 1.582e+01 7.962e+01 4.535e+02 8.864e-02 1.256e-01 1.201e-01 3.922e-02
 2.576e-01 7.018e-02]
[ 1.74409898  0.08404188  1.75250123  1.74655559  1.19418369  0.85530933
  1.62887018  2.02911184 -0.32259372 -0.32037375  0.11294252 -0.37299825
  0.08381117  0.31683111  0.67686775  0.10311192  0.84457344  1.20300252
  0.4603921   0.0782878   1.22281394 -0.07534795  1.18970895  1.11433115
  1.49103493  0.26135266  1.30546961  1.58979215  0.2021204  -0.18109876]
[-0.79835776 -1.32142888 -0.79674216 -0.69612431 -1.53650053 -0.80149195
 -0.60264801 -0.83000889 -0.40689723 -0.27150197 -0.00963789 -0.9474804
  0.09198759 -0.23810618 -0.65416946  0.33229233  0.13519555  0.26815273
  1.5255505   0.47601782 -0.76034388 -1.72181103 -0.72505522 -0.62531216
 -1.85056107 -0.79888327 -0.67880692 -1.10535627 -0.45904259 -0.75112464]
```

After that, we will initialize model in Python with creating a class named “LogisticRegressionUsingGD” and the model parameters in the class as well. Also, we

will define some functions in the class such as sigmoid function, training function and predict function, for training the logistic regression later and predicting the testing dataset too. The accuracy function can check the accuracy score with the predicted values of the model and the true values. The code of various parts in the class are shown in below figures.

The code for initializing the model parameters:

```
class LogisticRegressionUsingGD:
    def __init__(self, eta, n_iterations,lam):
        self.w = np.zeros(30)
        self.b = 0
        self.eta = eta
        self.n_iterations = n_iterations
        self.lam = lam
```

The code for defining the sigmoid function:

```
    def sigmoid(self, x):
        return 1.0/(1+np.exp(-x))
```

The code for defining the training function:

```
    def fit(self,x,y):
        itr = 0
        row, column = np.shape(x)
        print('number of samples:', row)
        while itr <= self.n_iterations:
            fx = np.dot(self.w, x.T) + self.b
            hx = self.sigmoid(fx)
            t = (hx-y)


            s = np.dot(x.T, t)


            gradient_w = s/row * self.eta
            gradient_b = np.sum(t)/row * self.eta
            self.w = self.w * (1 - self.lam * self.eta / row) - gradient_w
            self.b -= gradient_b
            itr += 1
```

The code for defining the predict function:

```
    def predict(self, x):
        fx = np.dot(self.w, x.T) + self.b
        hx = self.sigmoid(fx)
        ypt = []
        for i in hx:
            if i >= 0.5:
                ypt.append(1)
            else:
                ypt.append(0)

        return ypt
```

The code for defining the accuracy function:

```
    def accuracy(y_true, y_pred):
        accurary = np.sum(y_true == y_pred) / len(y_true)
        return accuracy
```

Fourth, we will define the main function. In order to prevent the problem of overfitting, we will build the logistic regression gradient descent algorithm with using regularization this time. We will also train the model with input parameters' values. It will include 1000 iterations, 1.2 as the learning rate and 3 sets of lambda to train the model. We can see that when setting lambda to 0, which is exactly the same situation without regularization, some of the magnitudes of coefficient are quite large. Since we want to decrease the coefficient magnitudes by regularization, it will show that lambda equals 10 or 100 comparing with lambda equals 0.

```
if __name__ == '__main__':


    theta = []


    print("lambda = 0")
    model = LogisticRegressionUsingGD(eta=1.2, n_iterations=1000,lam=0)
    model.fit(x_train_s,y_train)
    print(model.getB())
    print('Theta 1-30:')
    print(model.getW())
    theta.append(model.getW())
    print("-------------")
    print("lambda = 10")
    model = LogisticRegressionUsingGD(eta=1.2, n_iterations=1000,lam=10)
    model.fit(x_train_s,y_train)
    print(model.getB())
    print('Theta 1-30:')
    print(model.getW())
    theta.append(model.getW())
    print("-------------")
    print("lambda = 100")
    model = LogisticRegressionUsingGD(eta=1.2, n_iterations=1000,lam=100)
    model.fit(x_train_s,y_train)
    print(model.getB())
    print('Theta 1-30:')
    print(model.getW())
    theta.append(model.getW())
    print("-------------")
```

Fifth, we will use cross validation to split the given training dataset into 5 subsets of new training and testing dataset which have true value. Since it is a 5-folds cross validation, it will be split with different 100 rows of the given training dataset every time as the testing dataset and the remaining 400 rows of the given training dataset will be new training dataset. For example, it will be the first split for the first 100 rows as the testing data and the remaining data as the training data for both "x_train_s" and "y_train" datasets. After that, it will split the row 100 to row 199 as the testing data and the rest of the data as training data. It will be split until last 100 rows as the testing data and the remaining part as the training data, which means that we split five sets training and testing datasets of both given "x_train_s" and "y_train" dataset.

```
#Splitting first 100 rows as the testing data
xtrain0_99 = x_train_s[100:500, :]

xtest0_99 = x_train_s[:100, :]


ytrain0_99 = y_train[100:500, ]

ytest0_99 = y_train[:100, ]



#Splitting row 100 to row 199 as the testing data
xtrain100_199_1 = x_train_s[:100, :]
xtrain100_199_2 = x_train_s[200:500, :]

xtrain100_199 = np.concatenate((xtrain100_199_1, xtrain100_199_2))

xtest100_199 = x_train_s[100:200, :]


ytrain100_199_1 = y_train[:100, ]
ytrain100_199_2 = y_train[200:500, ]

ytrain100_199 = np.concatenate((ytrain100_199_1, ytrain100_199_2))

ytest100_199 = y_train[100:200, ]
```

```
#Splitting row 200 to row 299 as the testing data
xtrain200_299_1 = x_train_s[:200, :]
xtrain200_299_2 = x_train_s[300:500, :]

xtrain200_299 = np.concatenate((xtrain200_299_1, xtrain200_299_2))

xtest200_299 = x_train_s[200:300, :]


ytrain200_299_1 = y_train[:200, ]
ytrain200_299_2 = y_train[300:500, ]

ytrain200_299 = np.concatenate((ytrain200_299_1, ytrain200_299_2))

ytest200_299 = y_train[200:300, ]



#Splitting row 300 to row 399 as the testing data
xtrain300_399_1 = x_train_s[:300, :]
xtrain300_399_2 = x_train_s[400:500, :]

xtrain300_399 = np.concatenate((xtrain300_399_1, xtrain300_399_2))

xtest300_399 = x_train_s[300:400, :]


ytrain300_399_1 = y_train[:300, ]
ytrain300_399_2 = y_train[400:500, ]

ytrain300_399 = np.concatenate((ytrain300_399_1, ytrain300_399_2))

ytest300_399 = y_train[300:400, ]
```

```
#Splitting last 100 rows as the testing data
xtrain400_499 = x_train_s[0:400, :]

xtest400_499 = x_train_s[400:500, :]


ytrain400_499 = y_train[0:400, ]

ytest400_499 = y_train[400:500, ]
```

Then, 5 sets of new training dataset will be used to proceed the model fitting and model evaluation after building model with lambda equals 100. The number of samples used to fit for every time is 400. At this time, theta 0 to theta 30 and the performance of each model can be figured out. We can calculate the median of accuracy of five models and find out the model we will use. The code for model and model evaluation is shown in below figures.

```
model = LogisticRegressionUsingGD(eta=1.2, n_iterations=100,lam=100)

accu = []

print('\n')
print('Information about 1st set of split datasets')
#Fitting model with first set of split datasets
model.fit(xtrain0_99,ytrain0_99)

print("-------------")
print('Theta 0:')
print(model.getB())
print('Theta 1-30:')
print(model.getW())
print("-------------")

# Evaluating model with first set of split datasets
acc= accuracy_score(ytest0_99, model.predict(xtest0_99))
accu.append(acc)
print('Accuracy of testing dataset:',acc)
print('\n')
target_names = ['class 0', 'class 1']
print(classification_report(ytest, model.predict(xtest), target_names=target_names))
print("_____________________________________________________________________")
print('\n')
```

```
print('Information about 2nd set of split datasets')
#Fitting model with second set of split datasets
model.fit(xtrain100_199,ytrain100_199)

print("-------------")
print('Theta 0:')
print(model.getB())
print('Theta 1-30:')
print(model.getW())
print("-------------")

# Evaluating model with second set of split datasets
acc= accuracy_score(ytest100_199, model.predict(xtest100_199))
accu.append(acc)
print('Accuracy of testing dataset:',acc)
print('\n')
target_names = ['class 0', 'class 1']
print(classification_report(ytest, model.predict(xtest), target_names=target_names))
print("_____________________________________________________________________")
print('\n')
```

```
print('Information about 3rd set of split datasets')
#Fitting model with third set of split datasets
model.fit(xtrain200_299,ytrain200_299)

print("-------------")
print('Theta 0:')
print(model.getB())
print('Theta 1-30:')
print(model.getW())
print("-------------")

# Evaluating model with third set of split datasets
acc= accuracy_score(ytest200_299, model.predict(xtest200_299))
accu.append(acc)
print('Accuracy of testing dataset:',acc)
print('\n')
target_names = ['class 0', 'class 1']
print(classification_report(ytest, model.predict(xtest), target_names=target_names))
print("_____________________________________________________________________")
print('\n')
```

```
print('Information about 4th set of split datasets')
#Fitting model with fourth set of split datasets
model.fit(xtrain300_399,ytrain300_399)

print("-------------")
print('Theta 0:')
print(model.getB())
print('Theta 1-30:')
print(model.getW())
print("-------------")

# Evaluating model with fourth set of split datasets
acc= accuracy_score(ytest300_399, model.predict(xtest300_399))
accu.append(acc)
print('Accuracy of testing dataset:',acc)
print('\n')
target_names = ['class 0', 'class 1']
print(classification_report(ytest, model.predict(xtest), target_names=target_names))
print("_____________________________________________________________________")
print('\n')
```

```
print('Information about 5th set of split datasets')
#Fitting model with fifth set of split datasets
model.fit(xtrain400_499,ytrain400_499)

print("-------------")
print('Theta 0:')
print(model.getB())
print('Theta 1-30:')
print(model.getW())
print("-------------")

#Evaluating model with fifth set of split datasets
acc= accuracy_score(ytest400_499, model.predict(xtest400_499))
accu.append(acc)
print('Accuracy of testing dataset:',acc)
print('\n')
target_names = ['class 0', 'class 1']
print(classification_report(ytest, model.predict(xtest), target_names=target_names))
```

```
print("_____________________________________________________________________")
print('\n')
print('Five sets of accuaracy:')
print(accu)
print('The median of five sets of accuaracy:')
print(statistics.median(accu))
```

Finally, we will predict the labels of testing data after choosing fitted model. The code for prediction is shown below.

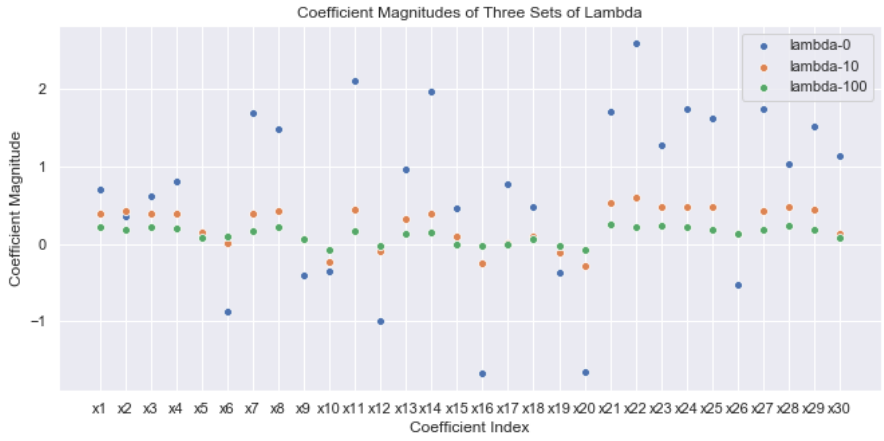
```
model = LogisticRegressionUsingGD(eta=1.2, n_iterations=100,lam=100)

#Fitting model with second set of split datasets
model.fit(xtrain100_199,ytrain100_199)

print("-------------")
print('Theta 0:')
print(model.getB())
print('Theta 1-30:')
print(model.getW())
print("-------------")

#Prediction
ypred = model.predict(x_test_s)
print(ypred)
temp = {'ypred': ypred}
ypreddf = pd.DataFrame(temp)
ypreddf.to_csv(r'Documents\COMP 4434\ypred', index=False)
```

```
#Prediction
ypred = model.predict(x_test_s)
print(ypred)
temp = {'ypred': ypred}
ypreddf = pd.DataFrame(temp)
ypreddf.to_csv(r'Documents\COMP 4434\ypred', index=False)

#plot
index = np.arange(1,51)
cotemp = { 'feature_label' : index,
                'ypred': ypred}

co = df_xy = pd.DataFrame(cotemp)

yp = sns.scatterplot(x="feature_label", y="ypred", data=co, label="ypred")

plt.title('Logistic Regression')
plt.xlabel('Information Attributes')
plt.ylabel('Diagnosis')
plt.legend()

plt.savefig('Logistic Regression.png')
```

4. Performance Evaluation and Discussion

The overfitting problem can be shown in the plot of “Coefficient Magnitudes of Three Sets of Lambda” and we can resolve the problem by implementing regularization. When lambda is equal to 10 and 100, the magnitudes of coefficient are reduced continuously. In the plot, when the value of lambda increased to 100, the magnitudes of some coefficient like x12, x16 and x20 can reduced by this method. Therefore, it can reduce the magnitudes by introducing parameter lambda to balance contribution of the features. It means that using the regularization method can decrease the error due to the high variance. The values of theta 0 to 30 with corresponding lambda and the plot presenting the coefficient magnitudes of three sets of lambda are shown below.

```
lambda = 0
number of samples: 500
-0.33973495698438816
Theta 1-30:
[ 0.69731468  0.35762162  0.62188205  0.80543277  0.10181815 -0.86818324
  1.68912998  1.48108264 -0.40022027 -0.34770545  2.10756093 -0.9913308
  0.97106091  1.95990963  0.45781333 -1.66889338  0.7802045   0.47045287
 -0.36892747 -1.65679915  1.71067027  2.58765471  1.28080114  1.74092981
  1.61715076 -0.52117666  1.7389909   1.03064505  1.52587938  1.12903228]
-------------
lambda = 10
number of samples: 500
-0.6235707098753966
Theta 1-30:
[ 0.39984005  0.43139745  0.38744771  0.38929817  0.15468263  0.01395307
  0.38695076  0.43398013  0.0552805  -0.23251026  0.44478242 -0.09263261
  0.32370695  0.3921899   0.09439334 -0.24344153  0.00941786  0.10084911
 -0.11109391 -0.28009186  0.52681073  0.59111923  0.48377275  0.48598265
  0.47147093  0.13241753  0.43026091  0.4698263   0.44476568  0.13682373]
-------------
lambda = 100
number of samples: 500
-0.6325606278088228
Theta 1-30:
[ 0.21520497  0.18589123  0.21307806  0.20337206  0.087681    0.09394291
  0.17243386  0.21630257  0.06116692 -0.08153651  0.15823866 -0.01899869
  0.13608873  0.15174179 -0.00170943 -0.02404244 -0.00698398  0.06153648
 -0.03002197 -0.08370501  0.2468494   0.22583582  0.23874548  0.2237374
  0.17536634  0.13465008  0.18893114  0.24204439  0.17562824  0.07290865]
```



With every fitted model of 5 sets of split datasets, we can use accuracy, precision, recall and F1-score to check the performance of the training model. We will test the

accuracy between the true label of testing dataset and the prediction of the testing dataset through the attributes with fitted model for 5 sets of split datasets. The five accuracy are quite high, and it shows that the fitted model is quite accurate for testing dataset. At last, we can calculate the median of 5 accuracy which is 0.96. Therefore, the second set of split datasets will be chosen to fit the model and to predict the labels of given testing dataset. In the fitted model with second set of split datasets, we can see that f-1 score in class 0 and class 1 perspective are 0.97 and 0.95. It is desirable for having high f-1 score of this model. We can also see that precision and recall are showing a good performance of the model. In the figures below, the information about 5 set of split datasets and the performance of corresponding model are shown.

```
Information about 1st set of split datasets
number of samples: 400
-------------
Theta 0:
-0.5843510235268464
Theta 1-30:
[ 0.19029532  0.17144519  0.1893318   0.18000657  0.08529825  0.09040348
  0.16005326  0.20563274  0.05216712 -0.06452087  0.14356749 -0.02318894
  0.12914974  0.1370032  -0.00104546 -0.01796991 -0.00611679  0.07158884
 -0.03686661 -0.07470825  0.21994768  0.21210406  0.21581979  0.19942058
  0.15887065  0.12960473  0.18102282  0.23290615  0.15954281  0.07978545]
-------------
Accuracy of testing dataset: 0.98


              precision    recall  f1-score   support

     class 0       0.92      1.00      0.96        70
     class 1       1.00      0.89      0.94        55

    accuracy                           0.95       125
   macro avg       0.96      0.95      0.95       125
weighted avg       0.96      0.95      0.95       125
```

```
Information about 2nd set of split datasets
number of samples: 400
-------------
Theta 0:
-0.5900797206345013
Theta 1-30:
[ 0.19749743  0.17125598  0.196589    0.18603872  0.07702297  0.09137209
  0.16257126  0.19801326  0.06988739 -0.07696934  0.11836635 -0.02021397
  0.10736306  0.12514267  0.00178118 -0.0059055  -0.0043648   0.0494639
 -0.01969632 -0.07583394  0.22555109  0.21163444  0.22202731  0.20425193
  0.16619611  0.13718507  0.18710131  0.22904125  0.17074535  0.0698596 ]
-------------
Accuracy of testing dataset: 0.96


              precision    recall  f1-score   support

     class 0       0.93      1.00      0.97        70
     class 1       1.00      0.91      0.95        55

    accuracy                           0.96       125
   macro avg       0.97      0.95      0.96       125
weighted avg       0.96      0.96      0.96       125
```

```
Information about 3rd set of split datasets
number of samples: 400
-------------
Theta 0:
-0.5971429603940824
Theta 1-30:
[ 0.19460163  0.16736948  0.1930731   0.18335512  0.08100465  0.09581925
  0.17247687  0.20313761  0.05820554 -0.06178002  0.15316502 -0.00871407
  0.12780969  0.14057392 -0.02504779 -0.02562129  0.01657426  0.05857114
 -0.02589092 -0.07231933  0.22655146  0.2025536   0.21758805  0.20376765
  0.16205815  0.1273508   0.18412216  0.22351867  0.1668268   0.06779533]
-------------
Accuracy of testing dataset: 0.97


              precision    recall  f1-score   support

     class 0       0.92      1.00      0.96        70
     class 1       1.00      0.89      0.94        55

    accuracy                           0.95       125
   macro avg       0.96      0.95      0.95       125
weighted avg       0.96      0.95      0.95       125

_____________________________________________________________________________

Information about 4th set of split datasets
number of samples: 400
-------------
Theta 0:
-0.6882516018227752
Theta 1-30:
[ 0.20364478  0.15704754  0.20190967  0.19327796  0.09089281  0.09469538
  0.1556302   0.2018258   0.06060316 -0.07186005  0.14735894 -0.01129622
  0.1238718   0.14328148 -0.01175329 -0.01136056 -0.01643382  0.0511968
 -0.02687092 -0.06082844  0.22895632  0.18921899  0.22005326  0.20898503
  0.1631654   0.12979216  0.16402951  0.21838045  0.15822507  0.0745929 ]
-------------
Accuracy of testing dataset: 0.91


              precision    recall  f1-score   support

     class 0       0.91      1.00      0.95        70
     class 1       1.00      0.87      0.93        55

    accuracy                           0.94       125
   macro avg       0.95      0.94      0.94       125
weighted avg       0.95      0.94      0.94       125

_____________________________________________________________________________
```

```
Information about 5th set of split datasets
number of samples: 400
-------------
Theta 0:
-0.7072408327021079
Theta 1-30:
[ 0.21202867  0.16693593  0.20901433  0.20034679  0.06930229  0.09192659
  0.1498214   0.19162978  0.05221422 -0.08004403  0.15590914 -0.02149183
  0.13751841  0.14788149  0.00598959 -0.01403185 -0.0030556   0.0663
 -0.02540662 -0.07184953  0.23375571  0.19234707  0.22553285  0.21338611
  0.13514858  0.1184871   0.15598222  0.21381434  0.13488918  0.05219248]
-------------
Accuracy of testing dataset: 0.92


              precision    recall  f1-score   support

     class 0       0.91      1.00      0.95        70
     class 1       1.00      0.87      0.93        55

    accuracy                           0.94       125
   macro avg       0.95      0.94      0.94       125
weighted avg       0.95      0.94      0.94       125

_____________________________________________________________________________


Five sets of accuaracy:
[0.98, 0.96, 0.97, 0.91, 0.92]
The median of five sets of accuaracy:
0.96
```

5. Conclude Discoveries

It is useful to use logistic regression to build the model if there is a classification problem and the dependent variable is split into classes. At this time, the dependent variable is a binary variable. We can see the results if the predicted value is 1 that it is representing the diagnosis of the lung cell is malignant and 0 is representing the lung cell is benign. Besides, the gradient descent in logistic regression can minimize the error by repeatedly updating the value of parameters.

Also, regularization is controlling the effect of various features and reducing the magnitudes of coefficient effectively. It has the notable results after increasing the parameter lambda and penalized term in the logistic regression model. It is good for balancing the effect and the importance for the information of the attributes.

By using the cross validation, it can help evaluate the fitted model that we can know about the performance of the fitted model and realize the split data whether it is suitable or not for model fitting.

Ultimately, the given testing dataset can predict the labels by the chosen fitted logistic regression model with regularization. 50 labels of the cell are predicted with 50 samples testing data. We can see that the predicted labels are 0 or 1 which are the labels of diagnosis. There are 15 labels which is labeled as 1, which means 15 cells are predicted as malignant, and 35 labels are labeled as 0 that 35 cells are predicted as benign. The predicted labels and the plot of showing the various labels are shown in the figure below.

```
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0,
0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1]
```