Assignment 02

Name:- Sk Fardeen Hossain

Roll No. :- 2021CSB023

G-Suite Id:- 2021csb023.sk@students.iiests.ac.in

Department:- Computer Science and Technology

Question 01

Download Cancer Wisconsin (Diagnostic) Data Set (already in the needed format). The dataset is used to recognize 2 types of cancer to be predicted (benign or malignant).

In [54]:
```
## Install and import the pandas library
!pip install pandas
```

```
Requirement already satisfied: pandas in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (from pandas) (2.0.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (from pandas)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (from python-dateutil>=2.8.2->p
andas) (1.16.0)
```

In [55]:
```
import pandas as pd

DATASET_PATH = "../ML_DRIVE/Assignment02/data.csv"
cancer_dataframe = pd.read_csv(DATASET_PATH)
print(cancer_dataframe.columns)
cancer_dataframe.head()
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

Out[55]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perim
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	

5 rows × 33 columns

In [56]:

```
## Pre-processing the data
cancer_dataframe = cancer_dataframe.drop('Unnamed: 32',axis=1) ### Dropping the column with null values
cancer_dataframe = cancer_dataframe.drop('id',axis=1) ### Dropping the id column
cancer_dataframe
```

Out[56]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	...	25.380
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	...	24.990
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	...	23.570
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	...	14.910
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	...	22.540
...	...	...	...	...	...	...	...	...	...	...	...	..
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	...	25.450
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	...	23.690
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	...	18.980
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	...	25.740
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	...	9.456

569 rows × 31 columns

Question 02

Implement Logistic Regression using scikit-learn package.

a. Use 'newton-cg', 'lbfgs', 'liblinear' solver for the regression model.

b. For each solver use 'l1', 'l2', 'none' penalty to train the model.

c. Split the dataset in to 80:10:10 percent (train : validation : test) (use seed = 5 for splitting).

d. Train the model initially with 80% training data and create a table for the coefficients of all the features.

e. Fine-tune the model with the remaining validation partition of the dataset (consisting of 10% of the original dataset) and create a table for the updated coefficients of all the features.

In [57]:
```
## Install the scikit-learn package
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (1.5.1)
Requirement already satisfied: numpy>=1.19.5 in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (from scikit-learn)
(2.0.1)
Requirement already satisfied: scipy>=1.6.0 in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (from scikit-learn) (1.1
4.0)
Requirement already satisfied: joblib>=1.2.0 in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (from scikit-learn)
(1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /home/fardeen/Desktop/MachineLearningLab/venv/lib/python3.12/site-packages (from scikit-learn)
(3.5.0)
```

In [58]:
```
## Separating the class variable from the dataset

import pandas as pd

y = cancer_dataframe[['diagnosis']]

print(y.head()) # -->  M implies Malignant B implies Benign
print(y.value_counts())
```

```
  diagnosis
0         M
1         M
2         M
3         M
4         M
diagnosis
B            357
M            212
Name: count, dtype: int64
```

In [59]:
```
## Rest of the features will be independent variables
X = cancer_dataframe.drop('diagnosis',axis=1)
X.head()
```

Out[59]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...

5 rows × 30 columns

Standardization of feature values to keep less bias on feature value thus maintaining uniformity and treating features on an equal scale

In [60]:

```
## Min-Max Scaler
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X = pd.DataFrame(
    columns=X.columns,
    data = scaler.fit_transform(X)
)

X.head()
```

Out[60]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...
0	0.521037	0.022658	0.545989	0.363733	0.593753	0.792037	0.703140	0.731113	0.686364	0.605518	...
1	0.643144	0.272574	0.615783	0.501591	0.289880	0.181768	0.203608	0.348757	0.379798	0.141323	...
2	0.601496	0.390260	0.595743	0.449417	0.514309	0.431017	0.462512	0.635686	0.509596	0.211247	...
3	0.210090	0.360839	0.233501	0.102906	0.811321	0.811361	0.565604	0.522863	0.776263	1.000000	...
4	0.629893	0.156578	0.630986	0.489290	0.430351	0.347893	0.463918	0.518390	0.378283	0.186816	...

5 rows × 30 columns

Splitting the dataset into training, testing and validation data

In [61]:

```
from sklearn.model_selection import train_test_split

X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.2, random_state=5)

X_valid, X_test, y_valid, y_test = train_test_split(X_rem, y_rem, train_size=0.5, random_state=5)

# print(X_train.info())
```

```python
# print(X_test.info())
# print(X_valid.info())
# print(y_train.info())
# print(y_test.info())
# print(y_valid.info())
```

In [70]:
```python
## Utility Function to create and vary the Logistic Regression model

import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score
'''
    @brief:- Function that creates a logisitic regression model and returns the accuracy and coefficient of all the features

    @params:- X_train  -> training data independent variable
              y_train  -> training data class variable
              X_valid  -> validation data independent variable
              y_valid  -> validation data class variable
              solver   -> solver used for fitting the model and increase convergence speed
              penalty  -> l1/l2 regularization to prevent over-fitting
              max_iter -> Max iterations for fitting the model

    @return:- Model coefficients(before and after finetuning), accuracy score, precision score, recall score, f1score


'''

def utilLogRegInfo(
    X_train:         'pd.Dataframe',
    y_train:         'pd.Dataframe',
    X_valid:         'pd.Dataframe',
    y_valid:         'pd.Dataframe',
    X_test:          'pd.Dataframe',
    y_test:          'pd.Dataframe',
    solver:          'str',
    penalty:         'str'='none',
    inv_reg_strength:'float'=1.0,
    max_iter:        'int' =10000) -> 'list':

    if penalty=='none':
        model = LogisticRegression(
            solver = solver,
            max_iter= max_iter,
            penalty=None,
            C=inv_reg_strength
        ).fit(X_train,y_train['diagnosis'])
        coef_train = model.coef_
        y_pred_before_finetune = model.predict(X_valid)
        accuracy_before_fine_tune = accuracy_score(y_valid,y_pred_before_finetune)
        precision_before_fine_tune = precision_score(y_valid,y_pred_before_finetune,average='macro').item()
        recall_before_fine_tune = recall_score(y_valid,y_pred_before_finetune,average='macro').item()
        F1score_before_fine_tune = f1_score(y_valid,y_pred_before_finetune,average='macro').item()
```

```python
        model.fit(np.vstack([X_train,X_valid]),np.vstack([y_train,y_valid]))
        coef_finetuned = model.coef_
        y_pred_after_finetune = model.predict(X_test)
        accuracy_after_fine_tune = accuracy_score(y_test,y_pred_after_finetune)
        precision_after_fine_tune = precision_score(y_test,y_pred_after_finetune,average='macro').item()
        recall_after_fine_tune = recall_score(y_test,y_pred_after_finetune,average='macro').item()
        F1score_after_fine_tune = f1_score(y_test,y_pred_after_finetune,average='macro').item()

    else:
        model = LogisticRegression(
            solver = solver,
            penalty = penalty,
            max_iter= max_iter,
            C=inv_reg_strength
        ).fit(X_train,y_train['diagnosis'])
        coef_train = model.coef_
        y_pred_before_finetune = model.predict(X_valid)
        accuracy_before_fine_tune = accuracy_score(y_valid,y_pred_before_finetune)
        precision_before_fine_tune = precision_score(y_valid,y_pred_before_finetune,average='macro').item()
        recall_before_fine_tune = recall_score(y_valid,y_pred_before_finetune,average='macro').item()
        F1score_before_fine_tune = f1_score(y_valid,y_pred_before_finetune,average='macro').item()
        model.fit(np.vstack([X_train,X_valid]),np.vstack([y_train,y_valid]))
        coef_finetuned = model.coef_
        y_pred_after_finetune = model.predict(X_test)
        accuracy_after_fine_tune = accuracy_score(y_test,y_pred_after_finetune)
        precision_after_fine_tune = precision_score(y_test,y_pred_after_finetune,average='macro').item()
        recall_after_fine_tune = recall_score(y_test,y_pred_after_finetune,average='macro').item()
        F1score_after_fine_tune = f1_score(y_test,y_pred_after_finetune,average='macro').item()

    return [solver,penalty,inv_reg_strength,accuracy_before_fine_tune,precision_before_fine_tune,recall_before_fine_tune,
            F1score_before_fine_tune,accuracy_after_fine_tune,precision_after_fine_tune,recall_after_fine_tune,F1score_after_fine_tune]+\
            coef_train.tolist()[0] + coef_finetuned.tolist()[0]
```

In [71]:

```python
newton_cg_reg_l2 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='newton-cg',penalty='l2')
newton_cg_reg_none = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='newton-cg')
lbfgs_reg_l2 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='lbfgs',penalty='l2')
lbfgs_reg_none = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='lbfgs')
liblinear_reg_l1 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='liblinear',penalty='l1')
liblinear_reg_l2 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='liblinear',penalty='l2')

solvers_report = pd.DataFrame(
    columns= ['solver','penalty']+[f"{col} coeff." for col in X_train.columns] + [f"{col} finetuned coeff. " for col in X_train.columns],
    data = [newton_cg_reg_l2[:2]+newton_cg_reg_l2[11:],newton_cg_reg_none[:2]+newton_cg_reg_none[11:],
            lbfgs_reg_l2[:2]+lbfgs_reg_l2[11:],lbfgs_reg_none[:2]+lbfgs_reg_none[11:],
            liblinear_reg_l1[:2]+liblinear_reg_l1[11:],
            liblinear_reg_l2[:2]+liblinear_reg_l2[11:]]
)

solvers_report
```

Out[71]:

	solver	penalty	radius_mean coeff.	texture_mean coeff.	perimeter_mean coeff.	area_mean coeff.	smoothness_mean coeff.	compactness_mean coeff.	concavity_mean coeff.	concave points_mean coeff.	...	radius_worst finetuned coeff.	texture_w finetu co
0	newton-cg	l2	1.763553	1.586001	1.732097	1.497118	0.581991	0.393523	1.438618	2.084012	...	2.344268	2.345
1	newton-cg	none	-3.832666	-16.951656	-33.221431	-32.735619	22.313882	-74.121615	77.898319	58.902442	...	-112.911015	77.346
2	lbfgs	l2	1.760413	1.587441	1.728953	1.498279	0.581363	0.391197	1.437480	2.088940	...	2.347741	2.342
3	lbfgs	none	-107.628970	-251.288585	-134.686577	-6.218266	54.224016	-311.087162	279.523186	451.847827	...	43.177874	50.340
4	liblinear	l1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.964401	...	13.411096	5.432
5	liblinear	l2	0.732458	0.838540	0.795956	1.056228	-0.396246	0.377031	1.863024	2.470633	...	1.729162	1.622

6 rows × 62 columns

Question 03

For every solver vary the 'l1' penalty over the range (0.1, 0.25, 0.75, 0.9) and compare the coefficients of the features.

In [73]:

```
## Only liblinear and saga solver supports l1 penalty

# liblinear
liblinear_l1_penalty = pd.DataFrame(
    columns =  ['solver','penalty','inverse_regularization_strength'] + [f"{col}_coeff. " for col in X_train.columns] +
                [f"{col} finetuned coeff. " for col in X_train.columns],

    data = [ utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test, solver='liblinear',inv_reg_strength=penalty,penalty='l1')[:3]
            + utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test, solver='liblinear',inv_reg_strength=penalty,penalty='l1')[11:]

            for penalty in [0.1,0.25,0.75,0.9]
           ]
)

saga_l1_penalty = pd.DataFrame(
    columns = ['solver','penalty','inverse_regularization_strength'] + [f"{col}_coeff. " for col in X_train.columns] +
                [f"{col} finetuned coeff. " for col in X_train.columns],

    data = [ utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='saga',inv_reg_strength=penalty,penalty='l1')[:3] +
             utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test, solver='saga',inv_reg_strength=penalty,penalty='l1')[11:]

            for penalty in [0.1,0.25,0.75,0.9]
           ]
)

frames = [liblinear_l1_penalty,saga_l1_penalty]
```

```
l1_penalty_report = pd.concat(frames)

l1_penalty_report
```

Out[73]:

	solver	penalty	inverse_regularization_strength	radius_mean_coeff.	texture_mean_coeff.	perimeter_mean_coeff.	area_mean_coeff.	smoothness_mean_coeff.	compactness_m
0	liblinear	l1	0.10	0.0	0.0	0.0	0.0	0.0	
1	liblinear	l1	0.25	0.0	0.0	0.0	0.0	0.0	
2	liblinear	l1	0.75	0.0	0.0	0.0	0.0	0.0	
3	liblinear	l1	0.90	0.0	0.0	0.0	0.0	0.0	
0	saga	l1	0.10	0.0	0.0	0.0	0.0	0.0	
1	saga	l1	0.25	0.0	0.0	0.0	0.0	0.0	
2	saga	l1	0.75	0.0	0.0	0.0	0.0	0.0	
3	saga	l1	0.90	0.0	0.0	0.0	0.0	0.0	

8 rows × 63 columns

Question 04

Using the test split (of 10%) to show the accuracy, precision, recall, F1-score of the regression model.

a. Show the output for every possible combination of solver-penalty (newton- cg-l1, newton-cg-l2, newton-cg-none,...)

b. Show the output for both the situations: one before performing fine-tuning on the model (with validation data split) and one after performing the fine-tuning on the model (with validation split).

c. Comment on the improvement, if any.

Previously we have performed the comparative accuracy, recall, precision and f1-score analysis alongwith coefficients of independent variables for variated penalty on solvers like newton-cg, liblinear, lbfgs both before and after fine-tuning, we will here also perform the same analysis on all the possible solvers

In [74]:

```
newton_cg_reg_l2 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='newton-cg',penalty='l2')[:11]
newton_cg_reg_none = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='newton-cg')[:11]
lbfgs_reg_l2 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='lbfgs',penalty='l2')[:11]
lbfgs_reg_none = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='lbfgs')[:11]
liblinear_reg_l1 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='liblinear',penalty='l1')[:11]
liblinear_reg_l2 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='liblinear',penalty='l2')[:11]
sag_reg_l2 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='sag',penalty='l2')[:11]
sag_reg_none = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='sag')[:11]
saga_reg_l1 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='saga',penalty='l1')[:11]
saga_reg_l2 = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='saga',penalty='l2')[:11]
saga_reg_none = utilLogRegInfo(X_train,y_train,X_valid,y_valid,X_test,y_test,solver='saga')[:11]


final_report = pd.DataFrame(
```

```python
    columns= ['solver','penalty','inverse_regularization_strength','accuracy_before_finetuning','precision_before_finetuning',
              'recall_before_finetuning','F1_score_before_finetuning','accuracy_after_finetuning','precision_after_finetuning',
              'recall_after_finetuning','F1_score_after_finetuning'],
    data = [ newton_cg_reg_l2[:11],newton_cg_reg_none[:11],lbfgs_reg_l2[:11],lbfgs_reg_none[:11],liblinear_reg_l1[:11],
            liblinear_reg_l2[:11], sag_reg_l2[:11], sag_reg_none[:11], saga_reg_l1[:11], saga_reg_l2[:11], saga_reg_none[:11]]
)

final_report
```

Out[74]:

	solver	penalty	inverse_regularization_strength	accuracy_before_finetuning	precision_before_finetuning	recall_before_finetuning	F1_score_before_finetuning	accuracy_afte
0	newton-cg	l2	1.0	0.964912	0.968750	0.962963	0.964640	
1	newton-cg	none	1.0	0.964912	0.964815	0.964815	0.964815	
2	lbfgs	l2	1.0	0.964912	0.968750	0.962963	0.964640	
3	lbfgs	none	1.0	0.964912	0.964815	0.964815	0.964815	
4	liblinear	l1	1.0	0.964912	0.968750	0.962963	0.964640	
5	liblinear	l2	1.0	0.947368	0.954545	0.944444	0.946779	
6	sag	l2	1.0	0.964912	0.968750	0.962963	0.964640	
7	sag	none	1.0	0.964912	0.964815	0.964815	0.964815	
8	saga	l1	1.0	0.964912	0.968750	0.962963	0.964640	
9	saga	l2	1.0	0.964912	0.968750	0.962963	0.964640	
10	saga	none	1.0	0.964912	0.964815	0.964815	0.964815	

Improvements

1. Cross-Validation

```
K-Fold Cross-Validation: Using cross-validation to ensure our model is not overfitting to the training data and to get a better
estimate of its performance on unseen data.
```

2. Advanced Optimization Techniques

```
Grid Search or Random Search: Grid Search or Random Search to systematically search for the best hyperparameters.
Bayesian Optimization: Bayesian optimization for more efficient hyperparameter tuning.
```

3. Ensemble of Logistic Regression Models

```
Model Averaging: Training multiple Logistic Regression models with different hyperparameters and average their predictions to
reduce variance.
```

4. Regularization Techniques

Elastic Net: Combining L1 and L2 regularization to benefit from both techniques.

5. Dimensionality Reduction

Dimensionality Reduction: Using various techniques like PCA to focus more on significant features that aids in processing and overall accuracy.