Project1

November 18, 2023

1 Project 1

1.1 Author

Name: Yelisetty Karthikeya S M Roll No.: 21CS30060

1.2 Imports

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix
from tabulate import tabulate
```

1.3 Load the dataset

[2]:
```
data = pd.read_csv('breast_cancer_data.csv')
print(data.shape)
data.head().T # Transpose the data to see all the columns
```

```
(569, 32)
```

[2]:

	0	1	2	3	4
id	842302	842517	84300903	84348301	84358402
diagnosis	M	M	M	M	M
radius_mean	17.99	20.57	19.69	11.42	20.29
texture_mean	10.38	17.77	21.25	20.38	14.34
perimeter_mean	122.8	132.9	130.0	77.58	135.1
area_mean	1001.0	1326.0	1203.0	386.1	1297.0
smoothness_mean	0.1184	0.08474	0.1096	0.1425	0.1003
compactness_mean	0.2776	0.07864	0.1599	0.2839	0.1328
concavity_mean	0.3001	0.0869	0.1974	0.2414	0.198
concave points_mean	0.1471	0.07017	0.1279	0.1052	0.1043
symmetry_mean	0.2419	0.1812	0.2069	0.2597	0.1809
fractal_dimension_mean	0.07871	0.05667	0.05999	0.09744	0.05883
radius_se	1.095	0.5435	0.7456	0.4956	0.7572

texture_se	0.9053	0.7339	0.7869	1.156	0.7813
perimeter_se	8.589	3.398	4.585	3.445	5.438
area_se	153.4	74.08	94.03	27.23	94.44
smoothness_se	0.006399	0.005225	0.00615	0.00911	0.01149
compactness_se	0.04904	0.01308	0.04006	0.07458	0.02461
concavity_se	0.05373	0.0186	0.03832	0.05661	0.05688
concave points_se	0.01587	0.0134	0.02058	0.01867	0.01885
symmetry_se	0.03003	0.01389	0.0225	0.05963	0.01756
fractal_dimension_se	0.006193	0.003532	0.004571	0.009208	0.005115
radius_worst	25.38	24.99	23.57	14.91	22.54
texture_worst	17.33	23.41	25.53	26.5	16.67
perimeter_worst	184.6	158.8	152.5	98.87	152.2
area_worst	2019.0	1956.0	1709.0	567.7	1575.0
smoothness_worst	0.1622	0.1238	0.1444	0.2098	0.1374
compactness_worst	0.6656	0.1866	0.4245	0.8663	0.205
concavity_worst	0.7119	0.2416	0.4504	0.6869	0.4
concave points_worst	0.2654	0.186	0.243	0.2575	0.1625
symmetry_worst	0.4601	0.275	0.3613	0.6638	0.2364
fractal_dimension_worst	0.1189	0.08902	0.08758	0.173	0.07678

1.4 Null values

[3]:
```
# null value inspection
data.isnull().sum()
```

[3]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
```

```
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

1.5 Data preprocessing

[4]:
```
# Assuming the 'diagnosis' column contains the target variable
data['diagnosis'] = data['diagnosis'].map({'M': 1, 'B': 0})
```

[5]:
```
# Split data into features and target
X = data.drop(['diagnosis'], axis=1)
y = data['diagnosis']
```

1.6 Split data into training and testing sets

[6]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

1.7 Feature scaling

[7]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

1.8 Logistic Regression

[8]:
```
class LogisticRegression:
    def __init__(self, lr=0.001, epochs=1000, log=False, log_freq=100,␣
 ↪early_stopping=True, patience=10):
        self.lr = lr
        self.epochs = epochs
        self.log = log
        self.log_freq = log_freq
        self.early_stopping = early_stopping
        self.patience = patience
        self.weights = None
        self.losses = {"train": [], "val": []}

    def fit(self, X_train, y_train, X_val=None, y_val=None):
```

```python
        self.weights = np.zeros((X_train.shape[1], 1))
        m = X_train.shape[0]
        if y_train.shape[0] != m:
            raise ValueError("Number of samples in X and y do not match.")

        for i in range(self.epochs):
            # Calculating loss for the training dataset
            y_train_pred = self.hypothesis(X_train)
            self.losses["train"].append(self.loss(y_train, y_train_pred))

            # Update the weights using the loss and learning rate
            self.weights -= self.lr * \
                (1/m) * np.dot(X_train.T, (y_train_pred - y_train.to_numpy().reshape(-1, 1)))


            if X_val is not None and y_val is not None:
                # Calculating loss for the validation dataset
                y_val_pred = self.hypothesis(X_val)
                self.losses["val"].append(self.loss(y_val, y_val_pred))

            if self.losses["train"][i] is float('inf'):
                print(
                    f"The weights are diverging after {i + 1} epochs. Try reducing the learning rate.")
                break

            if self.log and i % self.log_freq == 0:
                if X_val is not None and y_val is not None:
                    print(
                        f"After {i + 1} epochs, training loss: {self.losses['train'][i]}, validation loss: {self.losses['val'][i]}")
                else:
                    print(
                        f"After {i + 1} epochs, training loss: {self.losses['train'][i]}")

            if self.early_stopping and i > self.patience:
                if X_val is not None and y_val is not None:
                    if self.losses["val"][i - self.patience] < self.losses["val"][i]:
                        print(f"Early stopping after {i + 1} epochs.")
                        break
                else:
                    if self.losses["train"][i - self.patience] < self.losses["train"][i]:
```

```
                        print(f"Early stopping after {i + 1} epochs.")
                        break

        if self.log:
            if X_val is not None and y_val is not None:
                print(
                    f"Final training loss: {self.losses['train'][-1]}, final␣
↪validation loss: {self.losses['val'][-1]}")
            else:
                print(f"Final training loss: {self.losses['train'][-1]}")

        return self

    def hypothesis(self, X):
        z = X.dot(self.weights)
        return self.sigmoid(z)

    def sigmoid(self, z):
        return 1/(1+np.exp(-np.array(z, dtype=np.float64)))

    def loss(self, y, y_pred):
        return -1 * (np.dot(y.T, np.log(y_pred)) + np.dot((1-y).T, np.log(1 -␣
↪y_pred))).item() / len(y)

    def predict(self, X, threshold=0.5):
        return self.hypothesis(X) >= threshold

    def print_loss(self):
        print(
            f"Final training loss: {self.losses['train'][-1]}, final validation␣
↪loss: {self.losses['val'][-1]}")
```

1.9 Model initialization and training

[9]:
```
logistic_model = LogisticRegression()
svm_model = SVC()
nn_model = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000)

logistic_model.fit(X_train, y_train)
svm_model.fit(X_train, y_train)
nn_model.fit(X_train, y_train)
```

[9]:
```
MLPClassifier(max_iter=1000)
```

1.10 Model evaluation

[10]:
```python
models = [logistic_model, svm_model, nn_model]
model_names = ['Logistic Regression', 'SVM', 'Neural Network']
model_evaluations = {}

for model, name in zip(models, model_names):
    y_pred = model.predict(X_test)
    report_dict = classification_report(y_test, y_pred, output_dict=True)
    matrix = confusion_matrix(y_test, y_pred)
    model_evaluations[name] = [report_dict['accuracy'], report_dict['macro avg']['precision'], report_dict['macro avg']['recall'], report_dict['macro avg']['f1-score']]

    print(f"{(' '+name+' '):=^60}")
    print(f"Accuracy: {report_dict['accuracy']:.6f}")
    print(f"Precision: {report_dict['macro avg']['precision']:.6f}")
    print(f"Recall: {report_dict['macro avg']['recall']:.6f}")
    print(f"F1 Score: {report_dict['macro avg']['f1-score']:.6f}")
    print("\nConfusion Matrix:")
    print(tabulate(
        matrix,
        tablefmt="grid",
        showindex=["Actual 0", "Actual 1"],
        headers=["", "Predicted 0", "Predicted 1"]
        ))
```

```
=================== Logistic Regression ====================
Accuracy: 0.973684
Precision: 0.970130
Recall: 0.974288
F1 Score: 0.972120

Confusion Matrix:
+----------+---------------+---------------+
|          |   Predicted 0 |   Predicted 1 |
+==========+===============+===============+
| Actual 0 |            69 |             2 |
+----------+---------------+---------------+
| Actual 1 |             1 |            42 |
+----------+---------------+---------------+
=========================== SVM ============================
Accuracy: 0.982456
Precision: 0.986301
Recall: 0.976744
F1 Score: 0.981151

Confusion Matrix:
```

```
+----------+---------------+---------------+
|          |   Predicted 0 |   Predicted 1 |
+==========+===============+===============+
| Actual 0 |            71 |             0 |
+----------+---------------+---------------+
| Actual 1 |             2 |            41 |
+----------+---------------+---------------+
====================== Neural Network ======================
Accuracy: 0.973684
Precision: 0.974206
Recall: 0.969702
F1 Score: 0.971863

Confusion Matrix:
+----------+---------------+---------------+
|          |   Predicted 0 |   Predicted 1 |
+==========+===============+===============+
| Actual 0 |            70 |             1 |
+----------+---------------+---------------+
| Actual 1 |             2 |            41 |
+----------+---------------+---------------+
```

[11]:

```
print("Overall Evaluation:")
print(tabulate(
    [[f"{j*100:.2f} %" for j in i] for i in model_evaluations.values()],
    tablefmt="grid",
    showindex=["Logistic Regression", "SVM", "Neural Network"],
    headers=["", "Accuracy", "Precision", "Recall", "F1 Score"]
    ))
```

```
Overall Evaluation:
+---------------------+------------+-------------+----------+------------+
|                     | Accuracy   | Precision   | Recall   | F1 Score   |
+=====================+============+=============+==========+============+
| Logistic Regression | 97.37 %    | 97.01 %     | 97.43 %  | 97.21 %    |
+---------------------+------------+-------------+----------+------------+
| SVM                 | 98.25 %    | 98.63 %     | 97.67 %  | 98.12 %    |
+---------------------+------------+-------------+----------+------------+
| Neural Network      | 97.37 %    | 97.42 %     | 96.97 %  | 97.19 %    |
+---------------------+------------+-------------+----------+------------+
```