Manuel Logistic Regression

May 21, 2024

[56]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

[63]:
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def sgd(X, y, batch_size=10, learning_rate=0.1, max_iterations=100):
    np.random.seed(0)
    num_features = X.shape[1]
    w = np.random.randn(num_features)
    n = len(y)
    for iteration in range(max_iterations):
        indices = np.random.permutation(n)
        for start in range(0, n, batch_size):
            end = min(start + batch_size, n)
            batch_indices = indices[start:end]
            X_batch = X[batch_indices]
            y_batch = y[batch_indices]

            z = np.dot(X_batch, w)
            y_hat = sigmoid(z)

            gradient = np.dot(X_batch.T, (y_hat - y_batch))

            w -= learning_rate * gradient

    return w
```

[77]:
```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
X = data.data
y = data.target
```

[108]:
```
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
# Check for missing values
```

```
numMissing = df.isnull().sum().sum()
print(numMissing)
# Standardize values
print(df.info())
scaler = StandardScaler()
X = scaler.fit_transform(X)
# Split
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.
 ↪2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.25, random_state=42)
```

```
0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   mean radius             569 non-null    float64
 1   mean texture            569 non-null    float64
 2   mean perimeter          569 non-null    float64
 3   mean area               569 non-null    float64
 4   mean smoothness         569 non-null    float64
 5   mean compactness        569 non-null    float64
 6   mean concavity          569 non-null    float64
 7   mean concave points     569 non-null    float64
 8   mean symmetry           569 non-null    float64
 9   mean fractal dimension  569 non-null    float64
 10  radius error            569 non-null    float64
 11  texture error           569 non-null    float64
 12  perimeter error         569 non-null    float64
 13  area error              569 non-null    float64
 14  smoothness error        569 non-null    float64
 15  compactness error       569 non-null    float64
 16  concavity error         569 non-null    float64
 17  concave points error    569 non-null    float64
 18  symmetry error          569 non-null    float64
 19  fractal dimension error 569 non-null    float64
 20  worst radius            569 non-null    float64
 21  worst texture           569 non-null    float64
 22  worst perimeter         569 non-null    float64
 23  worst area              569 non-null    float64
 24  worst smoothness        569 non-null    float64
 25  worst compactness       569 non-null    float64
 26  worst concavity         569 non-null    float64
 27  worst concave points    569 non-null    float64
 28  worst symmetry          569 non-null    float64
 29  worst fractal dimension 569 non-null    float64
```

```
 30  target                    569 non-null    int64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
None
```

[81]:
```
from collections import Counter
class_counts = Counter(y_train)
print("Size of classes in training set: ", class_counts)
```

```
Size of classes in training set:  Counter({1: 216, 0: 125})
```

[105]:
```
weights = sgd(X_train, y_train, batch_size=10, learning_rate=0.01,␣
 ↪max_iterations=100)
def predict(X, weights):
    z = np.dot(X, weights)
    return sigmoid(z)
```

[106]:
```
y_pred = (predict(X_test, weights) >= 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.
 ↪4f}, F1-score: {f1:.4f}")
```

```
Accuracy: 0.9737, Precision: 0.9857, Recall: 0.9718, F1-score: 0.9787
```

Summary:

Based on these findings, it is obvious that the model does a great job with over a 0.97 in accuracy, precision, recall, and f1-score. I separated my code based on each part and my code is pretty straightforward. The linear regression uses all the predicters which are all floats to predict either a one or a zero.