# Standardization

August 28, 2025

[18]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# 1. Load dataset
X, y = load_breast_cancer(return_X_y=True) #X is just a NumPy array, not a␣
 ↪DataFrame

# 2. Split into train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 3. Logistic Regression WITHOUT scaling
model_raw = LogisticRegression(max_iter=2000)
model_raw.fit(X_train, y_train)
y_pred_raw = model_raw.predict(X_test)

print("Accuracy WITHOUT Standardization:", accuracy_score(y_test, y_pred_raw))
```

```
Accuracy WITHOUT Standardization: 0.956140350877193

/Users/xyang/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

### 0.0.1 Why the Warning Appeared

Logistic Regression is trained without scaling.

Logistic Regression uses gradient descent (via the lbfgs solver by default).

Because the features are on very different scales, the optimization landscape is badly conditioned → the solver struggles to converge within the default iterations

[19]:
```
# 4. Apply StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Logistic Regression WITH scaling
model_scaled = LogisticRegression(max_iter=2000)
model_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = model_scaled.predict(X_test_scaled)

print("Accuracy WITH Standardization:", accuracy_score(y_test, y_pred_scaled))
```

```
Accuracy WITH Standardization: 0.9736842105263158
```

### 0.0.2 How Standardization Fixes It

Each feature is rescaled to mean = 0, variance = 1. Now gradients flow evenly across all features. The optimizer converges much faster. The warning disappears. Accuracy often improves.

[ ]:
```

```

## 0.1 Get the Columns

[20]:
```
data = load_breast_cancer()
print(data.feature_names)    # list of 30 feature names
print(data.target_names)     # ['malignant' 'benign']
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
['malignant' 'benign']
```

## 0.2 Convert to a Pandas DataFrame

[21]:
```
# Create DataFrame with column names
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

print(df.head())          # first 5 rows
```

```
print(df.columns)          # column names
```

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                 0.07871  …          17.33           184.60      2019.0
1                 0.05667  …          23.41           158.80      1956.0
2                 0.05999  …          25.53           152.50      1709.0
3                 0.09744  …          26.50            98.87       567.7
4                 0.05883  …          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0

[5 rows x 31 columns]
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension',
       'target'],
```

```
      dtype='object')
```

### 0.2.1 The Core Problem: Features on Very Different Scales

Looking at the data:

mean radius ~ 10–30 mean area ~ 100–2000 mean smoothness ~ 0.05–0.2

This means some features are hundreds of times larger in magnitude than others.

If we feed this directly into algorithms like Logistic Regression, SVM, or Neural Networks (which rely on gradient descent), the large features (like mean area) will dominate the learning process. Smaller features (like mean smoothness) contribute very little — even if they are important predictors.

[ ]: