Locally Weighted Linear Regression (LWLR) - Complete Breakdown

---

1. In-Depth and Specific Intuitive Understanding

Locally Weighted Linear Regression (**LWLR**) is an extension of standard linear regression that does not assume a **global** linear relationship. Instead, it fits a new linear regression model for **each query point** by giving more importance (higher weights) to nearby training points.

Key Idea

- In **standard linear regression**, all training points contribute **equally** to the model.
- In **LWLR**, closer points contribute **more**, and distant points contribute **less**, using a **weighting function**.

Thus, instead of finding a single global best-fit line, LWLR finds **many local best-fit lines**, making it highly flexible for **non-linear data**.

2. When LWLR is Used and When It Should Be Avoided

✅ When to Use It

- When the relationship between variables is **non-linear**, but locally linear.
- When we need **high flexibility** in predictions.
- When we have **small or medium datasets**, since LWLR requires per-query computation.

❌ When to Avoid It

- If the dataset is **large**, LWLR is slow since it must solve a separate regression problem for each query.
- If **data is sparse**, LWLR may not have enough local points for meaningful regression.
- If we need a **global interpretable model**, LWLR does not provide a single equation like standard linear regression.

3. When It Fails to Converge and How to Avoid That

When LWLR Fails

- **Choice of bandwidth parameter τ**:
  - If τ is **too large**, LWLR behaves like standard linear regression (all points have similar weights).
  - If τ is **too small**, LWLR overfits and becomes unstable.
- **Multicollinearity in local points**: Just like in standard regression, if features are highly correlated, the local model can be unstable.
- **Singular matrix issue**: If a local region does not have enough variation in the features, the matrix inversion fails.

How to Avoid Convergence Issues

✅ Tune τ carefully to balance local vs. global structure.
✅ Use **regularization (Ridge Regression)** if matrix inversion fails.
✅ Ensure **enough local data points** exist for each prediction.

When LWLR Always Converges

- If the dataset is well-behaved (no extreme outliers, sufficient local samples).
- If $X^T X$ is invertible for the local points.
- If a reasonable τ value is chosen.

4. Advantages and Disadvantages

Advantages

✅ Handles **non-linearity** well by adapting locally.

✅ Requires **no feature transformation** (like polynomial features in standard regression).

✅ More **robust to irrelevant global patterns** since it focuses only on local trends.

Disadvantages

❌ **Computationally expensive**, requiring a separate regression for each query point.

❌ **Not interpretable** since it does not provide a single equation.

❌ **Sensitive to bandwidth (τ)**, requiring tuning.

5. Intuitive Algorithm / Pseudo Code

1. **For each query point** x_{query}:
   - Compute **weights** for all training points using:

     $$w_i = \exp\left(-\frac{(x_i - x_{query})^2}{2\tau^2}\right)$$

   - Construct a **diagonal weight matrix** W.
   - Solve the **weighted normal equation**:

     $$w = (X^T W X)^{-1} X^T W y$$

   - Predict:

     $$\hat{y}_{query} = x_{query} w$$

2. Repeat for **every query point**.

6. Mathematical and Logical Breakdown

Weighted Loss Function

Instead of minimizing **Mean Squared Error (MSE)**:

$$J(w) = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

LWLR minimizes a **weighted** loss function:

$$J(w) = \sum_{i=1}^{m} w_i (y_i - \hat{y}_i)^2$$

where w_i is a **weight function** that determines importance of each training point.

Weight Function (Gaussian Kernel)

$$w_i = \exp\left(-\frac{(x_i - x_{query})^2}{2\tau^2}\right)$$

where:

- x_i is a training sample.
- x_{query} is the query point.
- τ (bandwidth) controls **locality** (smaller τ → more local).

Normal Equation for LWLR

Once we compute the weight matrix W, the optimal parameters are obtained using:

$$w = (X^T W X)^{-1} X^T W y$$

where:

- X is the feature matrix.
- W is the **diagonal matrix of weights**.
- y is the output vector.

Prediction for a New Query Point

Once w is computed, prediction is:

$$\hat{y}_{query} = x_{query} w$$

7. Manual Implementation in Python

```python
import numpy as np

class LocallyWeightedLinearRegression:
    def __init__(self, tau=1.0):
        self.tau = tau  # Bandwidth parameter

    def _compute_weights(self, X, query_point):
        """Compute weights for training points based on query point."""
        m = X.shape[0]
        W = np.zeros((m, m))

        for i in range(m):
            W[i, i] = np.exp(-np.sum((X[i] - query_point) ** 2) / (2 * self.tau ** 2))

        return W

    def predict(self, X_train, y_train, X_query):
        """Predict values for each query point."""
        m, n = X_train.shape
        X_query = np.hstack([np.ones((X_query.shape[0], 1)), X_query])  # Add bias term
        X_train = np.hstack([np.ones((m, 1)), X_train])  # Add bias term

        y_pred = np.zeros(X_query.shape[0])

        for i in range(X_query.shape[0]):
            W = self._compute_weights(X_train, X_query[i])
            theta = np.linalg.pinv(X_train.T @ W @ X_train) @ (X_train.T @ W @ y_train)
            y_pred[i] = X_query[i] @ theta

        return y_pred

```

8. Implementation Using Scikit-Learn Approximation

Scikit-Learn does **not** have LWLR directly, but **KNN Regression with distance weighting** provides a similar effect.

python

```python
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression

# Generate dataset
X, y = make_regression(n_samples=100, n_features=1, noise=10, random_state=42)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# KNN Regression as LWLR Approximation (weights='distance' mimics LWLR)
model = KNeighborsRegressor(n_neighbors=10, weights='distance')
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Print first few predictions
print("Predictions:", y_pred[:5])
```

Final Summary

- **LWLR is a non-parametric regression** method that gives more importance to nearby points.
- It **adapts to non-linearity** without needing polynomial features.
- It does **not produce a global equation**, making it computationally expensive.
- **It is best for small datasets where local trends are important**.
- **It should be avoided for large datasets due to high computational cost**.

Comparison to Standard Linear Regression

Feature	Standard Linear Regression	LWLR
Global vs. Local	Global model (one equation)	Local models (different for each query)
Computational Complexity	Low	High
Handles Non-Linearity?	❌ No	✅ Yes
Interpretability	✅ Yes	❌ No
Speed on Large Datasets	✅ Fast	❌ Slow