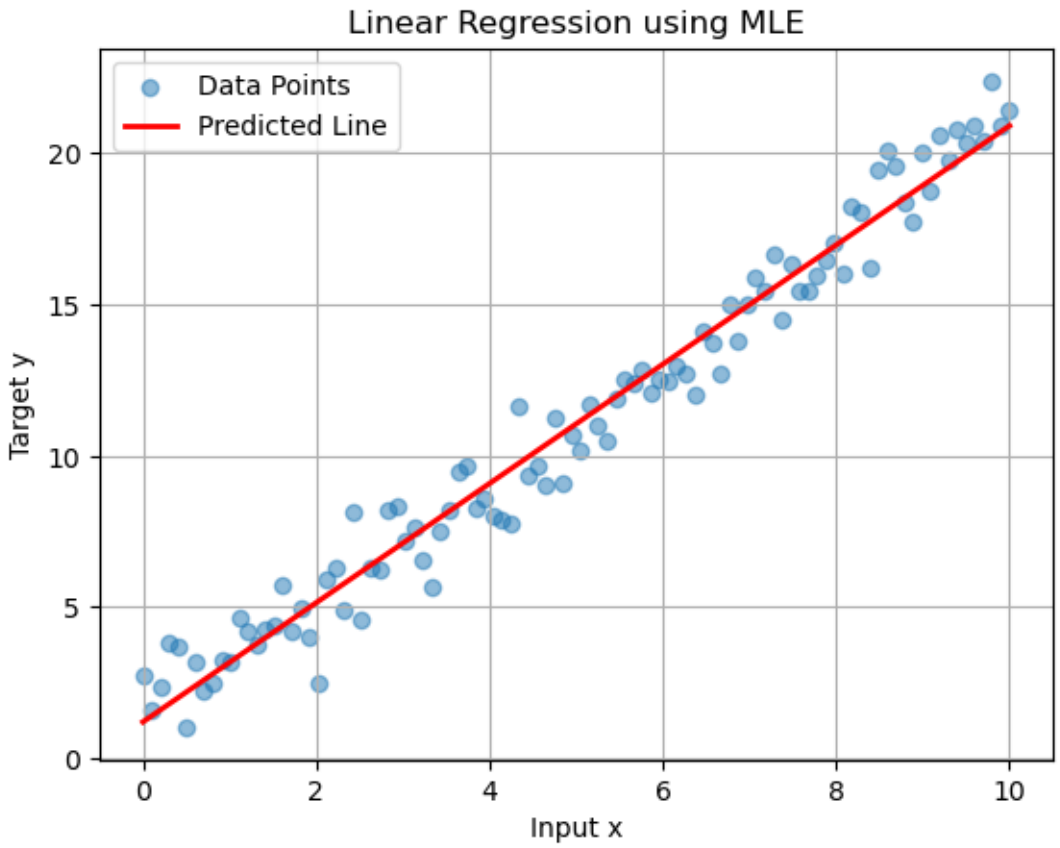
assn3-sagar-gupta-24-14-12-1

October 28, 2024

Assn3_Sagar Gupta_24-14-12

Q1

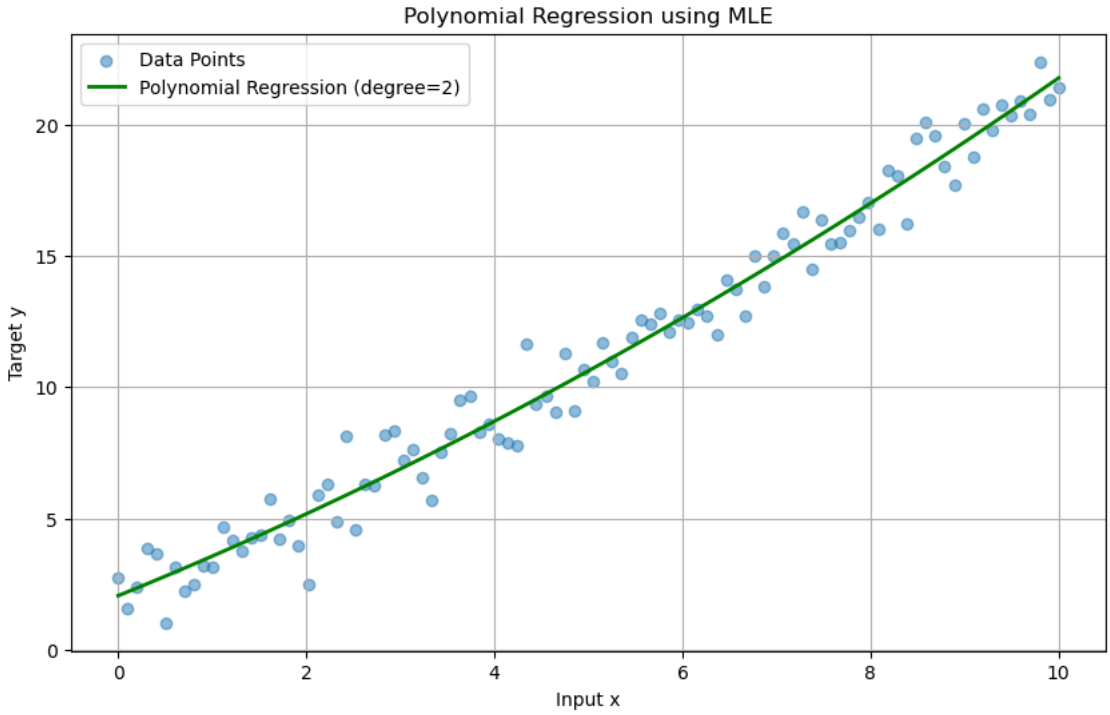
[24]:
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
x = np.linspace(0, 10, 100)
true_slope = 2
true_intercept = 1
noise = np.random.normal(0, 1, size=x.shape)
y = true_slope * x + true_intercept + noise

def compute_mle(x, y):
    X = np.vstack((np.ones_like(x), x)).T
    theta = np.linalg.inv(X.T .dot(X)) .dot (X.T .dot(y))
    return theta

theta = compute_mle(x, y)
y_hat = (theta[0] + theta[1] * x)

plt.scatter(x, y, label='Data Points', alpha=0.5)
plt.plot(x, y_hat, color='red', label='Predicted Line', linewidth=2)
plt.title('Linear Regression using MLE')
plt.xlabel('Input x')
plt.ylabel('Target y')
plt.legend()
plt.grid()
plt.show()
```



Q2

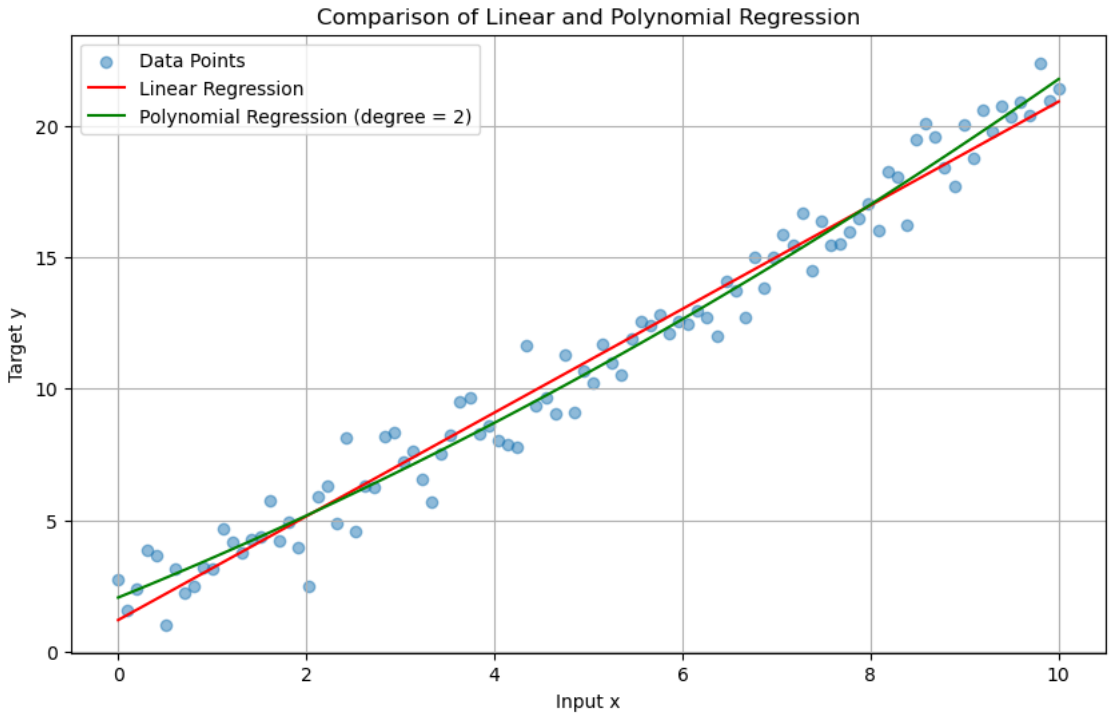
[27]:
```
np.random.seed(0)
x = np.linspace(0, 10, 100)
true_slope = 2
true_intercept = 1
noise = np.random.normal(0, 1, size=x.shape)
y = true_slope * x + true_intercept + noise

def create_polynomial_features(x, degree):
    return np.vstack([x**d for d in range(degree + 1)]).T

def compute_polynomial_mle(x, y, degree):
    X_poly = create_polynomial_features(x, degree)
    theta = np.linalg.inv(X_poly.T.dot(X_poly)).dot(X_poly.T.dot(y))
    return theta

degree = 2
theta_poly = compute_polynomial_mle(x, y, degree)
```

```
y_hat_poly = create_polynomial_features(x, degree).dot(theta_poly)

plt.figure(figsize=(10, 6))
plt.scatter(x, y, label='Data Points', alpha=0.5)
plt.plot(x, y_hat_poly, color='green', label='Polynomial Regression␣
 ↪(degree={})'.format(degree), linewidth=2)
plt.title('Polynomial Regression using MLE')
plt.xlabel('Input x')
plt.ylabel('Target y')
plt.legend()
plt.grid()
plt.show()
```



[20]:
```
np.random.seed(0)
x = np.linspace(0, 10, 100)
true_slope = 2
true_intercept = 1
noise = np.random.normal(0, 1, size=x.shape)
y = true_slope * x + true_intercept + noise

def create_polynomial_features(x, degree):
    return np.vstack([x**d for d in range(degree + 1)]).T

def compute_polynomial_mle(x, y, degree):
```

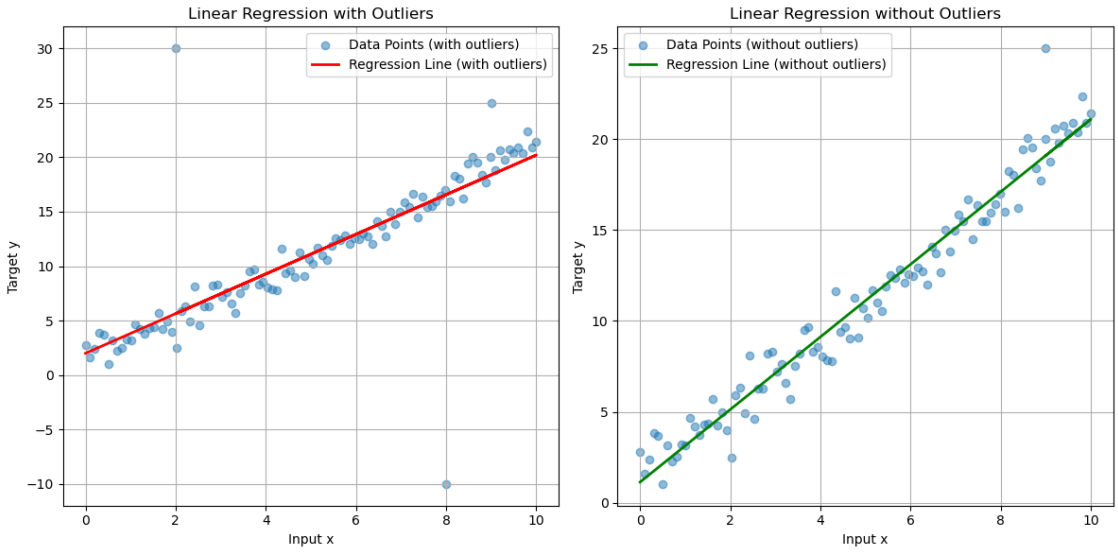
```
    X_poly = create_polynomial_features(x, degree)
    theta = np.linalg.inv(X_poly.T.dot(X_poly)).dot(X_poly.T.dot(y))
    return theta

def compute_linear_mle(x, y):
    return compute_polynomial_mle(x, y, degree=1)

theta_linear = compute_linear_mle(x, y)
y_hat_linear = create_polynomial_features(x, 1).dot(theta_linear)

degree = 2
theta_poly = compute_polynomial_mle(x, y, degree)
y_hat_poly = create_polynomial_features(x, degree).dot(theta_poly)

def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

mse_linear = mean_squared_error(y, y_hat_linear)
mse_poly = mean_squared_error(y, y_hat_poly)

plt.figure(figsize=(10, 6))
plt.scatter(x, y, label='Data Points', alpha=0.5)
plt.plot(x, y_hat_linear, color='red', label='Linear Regression')
plt.plot(x, y_hat_poly, color='green', label='Polynomial Regression (degree =␣
 ↪2)')
plt.title('Comparison of Linear and Polynomial Regression')
plt.xlabel('Input x')
plt.ylabel('Target y')
plt.legend()
plt.grid()
plt.show()
```



Q3

[32]:
```
np.random.seed(0)
x = np.linspace(0, 10, 100)
true_slope = 2
true_intercept = 1
noise = np.random.normal(0, 1, size=x.shape)
y = true_slope * x + true_intercept + noise

outlier_x = np.array([2, 8, 9])
outlier_y = np.array([30, -10, 25])
x = np.concatenate((x, outlier_x))
y = np.concatenate((y, outlier_y))

def compute_mle(x, y):
    X = np.vstack((np.ones_like(x), x)).T
    theta = np.linalg.inv(X.T.dot(X)).dot(X.T.dot(y))
    return theta

theta = compute_mle(x, y)
y_hat = (theta[0] + theta[1] * x)

def detect_outliers(x, y, threshold=2):
    residuals = y - (theta[0] + theta[1] * x)
```

```
    std_residuals = np.std(residuals)
    mean_residuals = np.mean(residuals)
    outliers = np.abs(residuals - mean_residuals) > threshold * std_residuals
    return outliers

outliers = detect_outliers(x, y)
x_cleaned = x[~outliers]
y_cleaned = y[~outliers]

theta_cleaned = compute_mle(x_cleaned, y_cleaned)
y_hat_cleaned = (theta_cleaned[0] + theta_cleaned[1] * x_cleaned)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(x, y, label='Data Points (with outliers)', alpha=0.5)
plt.plot(x, y_hat, color='red', label='Regression Line (with outliers)',␣
 ↪linewidth=2)
plt.title('Linear Regression with Outliers')
plt.xlabel('Input x')
plt.ylabel('Target y')
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)
plt.scatter(x_cleaned, y_cleaned, label='Data Points (without outliers)',␣
 ↪alpha=0.5)
plt.plot(x_cleaned, y_hat_cleaned, color='green', label='Regression Line␣
 ↪(without outliers)', linewidth=2)
plt.title('Linear Regression without Outliers')
plt.xlabel('Input x')
plt.ylabel('Target y')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()
```



[ ]: https://github.com/Frozen-icebar-007/Sagar-Gupta-_-24-14-12