09_regression_polynomial

May 7, 2025

1 Polynomial Regression

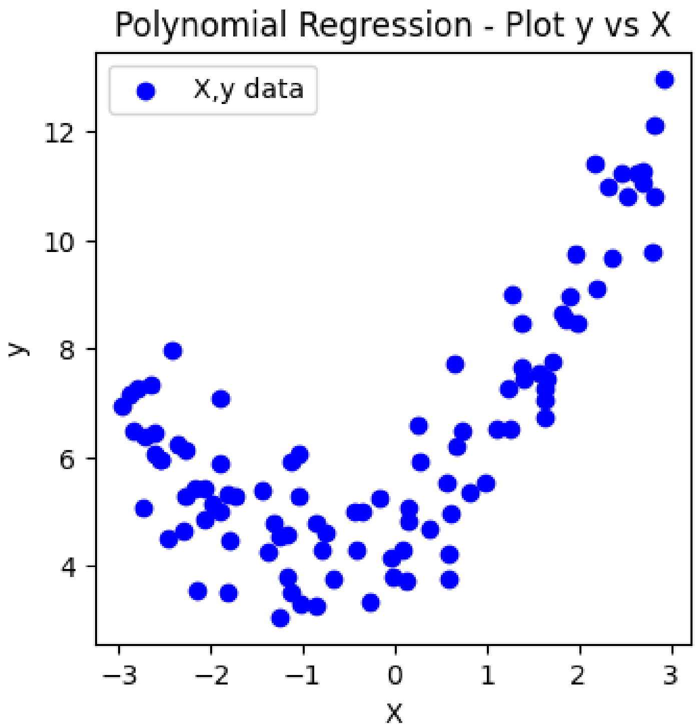
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
```

[ ]:
```
import os
user = os.getenv('USER')
os.chdir(f'/scratch/cd82/{user}/notebooks')
```

[2]:
```
# Generate non-linear data
N=100
np.random.seed(42)
X = 6 * np.random.rand(N, 1) - 3  # random  number between 0..1 which is scaled␣
 ↪by 6 and offset by -3
# Create our dependent data
y = 0.5 * X**2 + X + 5 + np.random.randn(N, 1)
```

Plot the data

[4]:
```
plt.figure(figsize=(4, 4))
plt.scatter(X, y,
    color='blue', label='X,y data')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Polynomial Regression - Plot y vs X')
plt.legend()
plt.show()
```



[5]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42)
```

Polynomial regression using sklearn The Scikit Learn library uses a `Pipeline` to create a pre-processing step before the regression fitting task.

[6]:
```
# Create polynomial regression model (degree=2)
poly_model = Pipeline([
    ("poly_features", PolynomialFeatures(degree=2, include_bias=False)),
        ("linear_regression", LinearRegression())
])
```

[7]:
```
# Train the model
poly_model.fit(X_train, y_train)
```

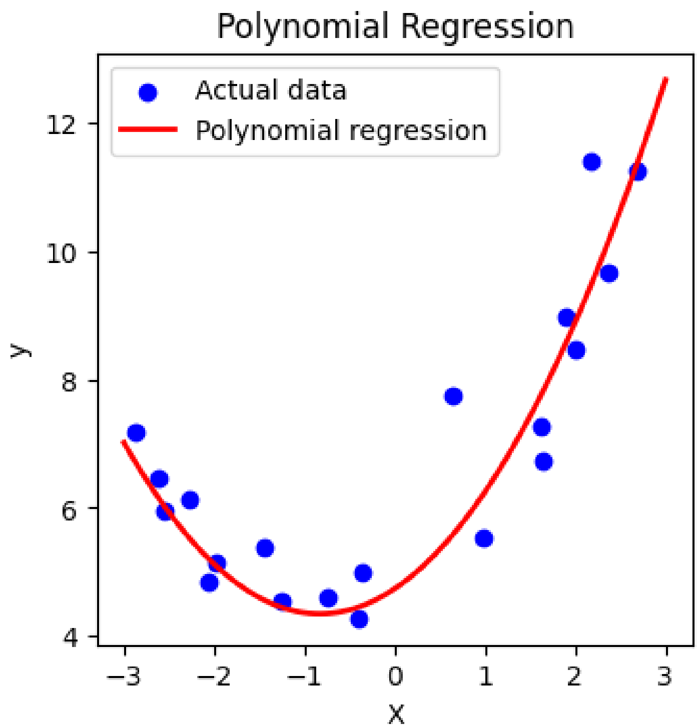
[7]:
```
Pipeline(steps=[('poly_features', PolynomialFeatures(include_bias=False)),
                ('linear_regression', LinearRegression())])
```

[8]:
```
# Make predictions
y_pred = poly_model.predict(X_test)
```

[9]:
```python
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R² Score: {r2}")
```

```
Mean Squared Error: 0.6358406072820812
R² Score: 0.8569223735172773
```

[12]:
```python
# Plot the results

X_plot = np.linspace(-3, 3, 100).reshape(-1, 1)
y_plot = poly_model.predict(X_plot)

plt.figure(figsize=(4, 4))
plt.scatter(X_test, y_test,
    color='blue', label='Actual data')
plt.plot(X_plot, y_plot, color='red',
    linewidth=2,
    label='Polynomial regression')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Polynomial Regression')
plt.legend()
plt.show()
```



Save the pipeline As the model looks good, we can save the pipeline so it can be run on new data.

[13]:
```python
import joblib

# Save the pipeline to a file
joblib.dump(poly_model, 'poly_model.pkl')

# Load the pipeline from the file
loaded_poly_pipeline = joblib.load('poly_model.pkl')

# Preprocess and predict on new data
new_data = X_test # Replace with your new dataset
predictions = loaded_poly_pipeline.predict(new_data)
```

Polynomial regression using `statsmodels` ref. https://ostwalprasad.github.io/ Polynomial-Regression-using-statsmodel.html

The `statsmodels` library does not have an automated method to add polynomial terms, so we can create our own function

[14]:
```python
import statsmodels.api as sm

# Create a function to add the squared term to the X data
def add_sqrd_column(X: np.ndarray, degree: int, index: int=0):
    # Select the column to modify
    modified_col = X[:, index]
    # modify the data to the desired power
    square_col = modified_col ** degree
    # Add the augmented column to the original matrix
    new_matrix = np.column_stack((X, modified_col))
    # return the new matrix
    return new_matrix

# Add the squared value of our data to our training matrix
X_train_p = add_sqrd_column(X_train, 2, 0)
X_test_p = add_sqrd_column(X_test, 2, 0)
# Add a constant to the model data (intercept)
X_train_p_int = sm.add_constant(X_train_p)
X_test_p_int = sm.add_constant(X_test_p)
```

[15]:
```python
# We can also use the same Scikit Learn class PolynomialFeatures
# It automatically adds an intercept column of 1's
from sklearn.preprocessing import PolynomialFeatures
polynomial_features= PolynomialFeatures(degree=3)
```

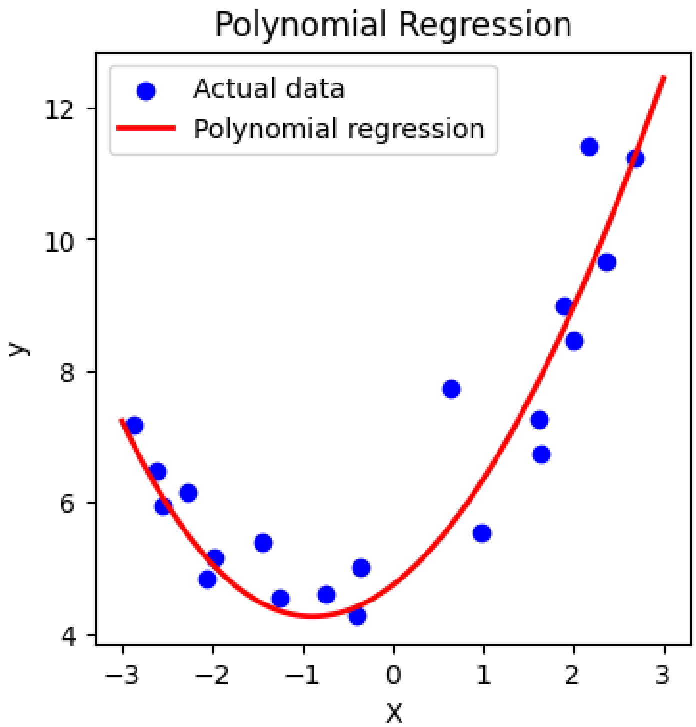
```
X_train_p_int = polynomial_features.fit_transform(X_train)
X_test_p_int = polynomial_features.fit_transform(X_test)


import statsmodels.api as sm
model_sm = sm.OLS(y_train, X_train_p_int).fit()
```

[16]:
```
# Make predictions
y_test_pred = model_sm.predict(X_test_p_int)
```

[17]:
```
# Evaluate the model
mse = mean_squared_error(y_test, y_test_pred)
r2 = r2_score(y_test, y_test_pred)

print(f"Mean Squared Error: {mse}")
print(f"R² Score: {r2}")
```

```
Mean Squared Error: 0.6420493386493977
R² Score: 0.8555252772366517
```

[18]:
```
# Plot the results
X_plot = np.linspace(-3, 3, 100).reshape(-1, 1)

X_plot_p_int = polynomial_features.fit_transform(X_plot)
# X_plot_p = add_sqrd_column(X_plot, 0)
# X_plot_p_int = sm.add_constant(X_plot_p)

y_plot = model_sm.predict(X_plot_p_int)

from statsmodels.sandbox.regression.predstd import wls_prediction_std
_, upper,lower = wls_prediction_std(model_sm)

# plt.plot(X_train_p_int, upper,'--',label="Upper") # confid. intrvl
# plt.plot(X_train_p_int, lower,':',label="lower")
# plt.legend(loc='upper left')
plt.figure(figsize=(4, 4))
plt.scatter(X_test, y_test,
    color='blue', label='Actual data')
plt.plot(X_plot, y_plot, color='red',
    linewidth=2,
    label='Polynomial regression')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Polynomial Regression')
plt.legend()
plt.show()
```



[19]:
```
model_sm.summary()
```

[19]:

Dep. Variable:	y	**R-squared:**	0.851
Model:	OLS	**Adj. R-squared:**	0.845
Method:	Least Squares	**F-statistic:**	144.7
Date:	Wed, 07 May 2025	**Prob (F-statistic):**	2.45e-31
Time:	15:45:33	**Log-Likelihood:**	-104.96
No. Observations:	80	**AIC:**	217.9
Df Residuals:	76	**BIC:**	227.5
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	4.7371	0.156	30.308	0.000	4.426	5.048
x1	1.0560	0.150	7.059	0.000	0.758	1.354
x2	0.5671	0.038	14.849	0.000	0.491	0.643
x3	-0.0207	0.025	-0.824	0.413	-0.071	0.029

Omnibus:	0.816	**Durbin-Watson:**	2.477
Prob(Omnibus):	0.665	**Jarque-Bera (JB):**	0.730
Skew:	0.229	**Prob(JB):**	0.694
Kurtosis:	2.901	**Cond. No.**	16.3

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[ ]:

```

```