Supervised Learning
– Regression and Deployment –

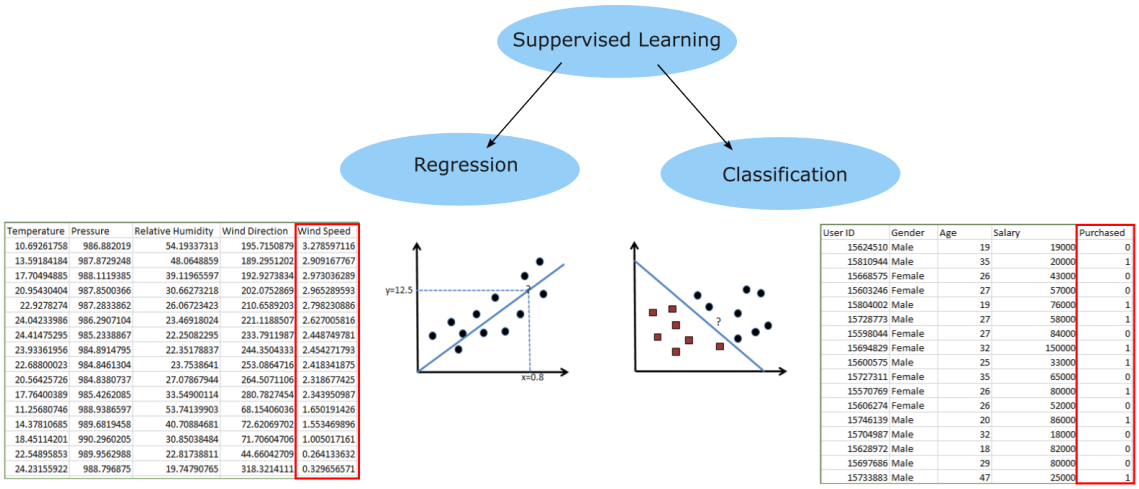
INT421 Applied Machine Learning (1-2023)

Niwan Wattanakitrungroj, Ph.D.

Contents

1. Supervised Learning Algorithms

1.1 Overview



Temperature	Pressure	Relative Humidity	Wind Direction	Wind Speed
10.69261758	986.882019	54.19337313	195.7150879	3.278597116
13.59184184	987.8729248	48.0648859	189.2951202	2.909167767
17.70494885	988.1119385	39.11965597	192.9273834	2.973036289
20.95430404	987.8500366	30.66273218	202.0752869	2.965289593
22.9278274	987.2833862	26.06723423	210.6589203	2.798230886
24.04233986	986.2907104	23.46918024	221.1188507	2.627005816
24.41475295	985.2338867	22.25082295	233.7911987	2.448749781
23.93361956	984.8914795	22.35178837	244.3504333	2.454271793
22.68800023	984.8461304	23.7538641	253.0864716	2.418341875
20.56425726	984.8380737	27.07867944	264.5071106	2.318677425
17.76400389	985.4262085	33.54900114	280.7827454	2.343950987
11.25680746	988.9386597	53.74139903	68.15406036	1.650191426
14.37810685	989.6819458	40.70884681	72.62069702	1.553469896
18.45114201	990.2960205	30.85038484	71.70604706	1.005017161
22.54895853	989.9562988	22.81738811	44.66042709	0.264133632
24.23155922	988.796875	19.74790765	318.3214111	0.329656571

User ID	Gender	Age	Salary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	1
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	1
15728773	Male	27	58000	1
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	1
15727311	Female	35	65000	0
15570769	Female	26	80000	1
15606274	Female	26	52000	0
15746139	Male	20	86000	1
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1

Two types of Supervised Learning:

Regression

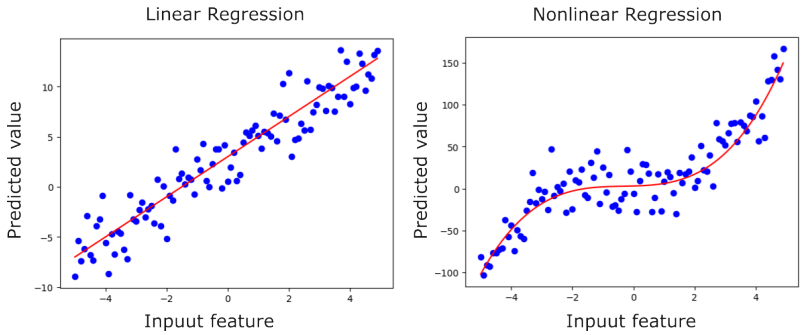
- Target is to predict continuous value
  - Predicting stock prices
  - Predicting income

Classification

- Target is to predict categorical value
  - binary (e.g., fraud, churn, credit risk)
  - multiclass (e.g., predicting credit ratings)

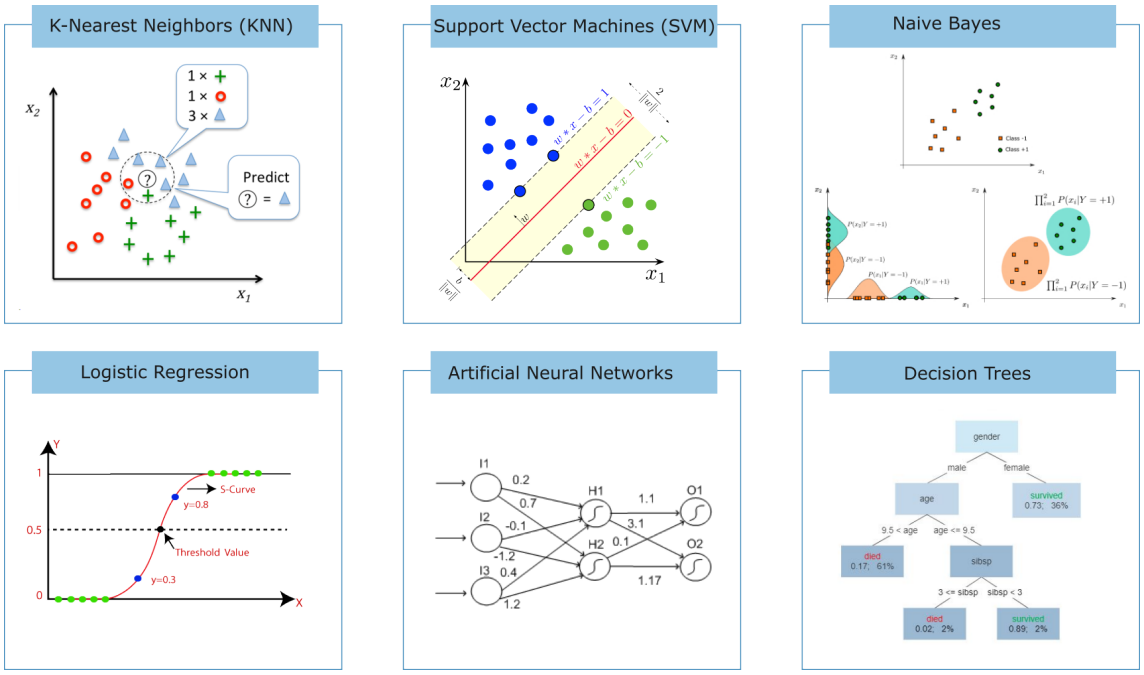
Regression Algorithms

- Linear Regression
- Nonlinear Regression



Classification Algorithms

- KNN: K-Nearest Neighbors
- SVM: Support Vector Machine
- Naive Bayes
- Logistic Regression
- ANN: Artificial Neural Networks
- Decision Tree



1.2 Linear Regression

Linear Regression is used for regression tasks, where the goal is to predict a continuous numeric output. It is commonly employed in scenarios such as predicting house prices, estimating sales revenue, or forecasting temperature.

Linear Equation: The core idea of Linear Regression is to represent the relationship between the target (y) and the features ($x_1, x_2, ..., x_n$) as a linear equation:

$$y = b_0 + b_1x_1 + b_2x_2 + \ldots + b_nx_n,$$

where

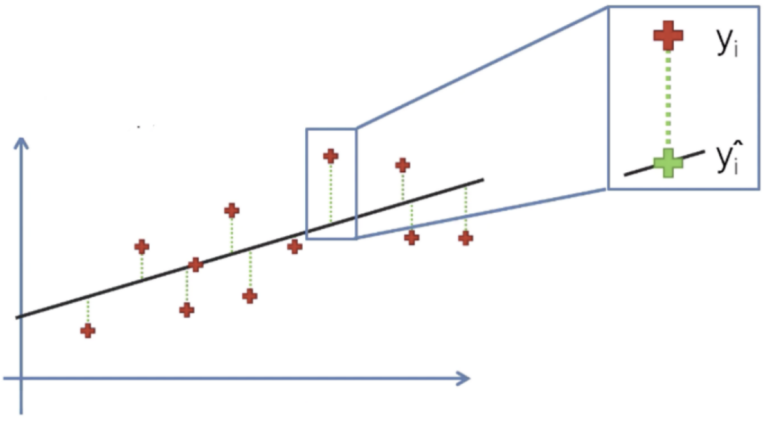
- y is the target.
- $x_1, x_2, \ldots, x_n$ are the features.
- $b_0, b_1, b_2, \ldots, b_n$ are coefficients (parameters) that need to be learned from the training data.

Training: The coefficients $b_0, b_1, b_2, \ldots, b_n$ are learned from a labeled training dataset. The learning process typically involves minimizing a cost function, often the mean squared error (MSE), to find the values of coefficients that best fit the training data.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where

- n: The total number of data points (samples)
- y_i: The actual or observed value for the i^{th} data point.
- $\hat{y}_i$: The predicted value for the i-th data point.



Training a linear regression model involves estimating the coefficients (parameters) of the linear equation that best fits the training data. In the case of simple linear regression (one feature), you have two coefficients to estimate: the intercept (b_0) and the slope (b_1). In multiple linear regression (multiple features), you have additional coefficients for each predictor variable.

The steps for training a linear regression model:

1. **Initialize Coefficients**: Start by initializing the coefficients ($b_0, b_1, b_2, ..., b_n$) to some initial values, often set to 0 or small random values.
2. **Define the Cost Function**: Choose a cost function that quantifies the difference between the predicted values and the actual target values. In linear regression, the common cost function is the Mean Squared Error (MSE):

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

3. **Gradient Descent** (Optimization): Gradient Descent is a common optimization algorithm used to minimize the cost function and find the optimal values of the coefficients.
   For each iteration, calculate the gradient (partial derivative) of the cost function with respect to each coefficient.
   Update the coefficients as follows:

$$b_j = b_j - \alpha\frac{\partial}{\partial b_j}MSE$$

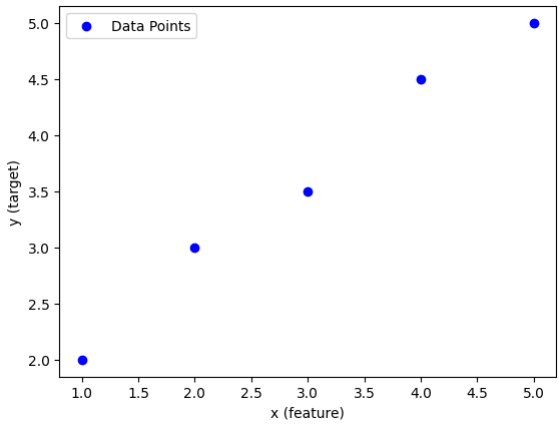
$$b_0 = b_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)$$

$$b_j = b_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)\cdot x_i, \quad \text{if} \quad j \neq 0$$

4. **Iterate:** Repeat the gradient descent process for a fixed number of iterations (epochs) or until the cost function converges to a minimum (i.e., the change in cost becomes very small).

Practice:
Training data:

x (feature)	y (target)
3	3.5
1	2
2	3
4	4.5
5	5



Initial Coefficients:
- $b_0 = 0$
- $b_1 = 0.5$

Learning Rate (α): 0.1 (a small positive value)

Iteration 1:

Step 1: Calculate Predicted Values

In each iteration, we calculate the predicted values $\hat{y}_i$ for each data point using the current coefficients.

x (feature)	y (target)	$\hat{y}$ (predicted)
3	3.5	
1	2	
2	3	
4	4.5	
5	5	

Step 2: Calculate Error (Residuals)

For each data point, calculate the error (residual) between the predicted value $\hat{y}_i$ and the actual target value y_i.

x (feature)	y (target)	$\hat{y}$ (predicted)	$e = \hat{y} - y$
3	3.5		
1	2		
2	3		
4	4.5		
5	5		

Step 3: Update Coefficients

Using the gradient descent update rules, update the coefficients b_0 and b_1:

$b_0 = b_0 - \alpha * \frac{1}{5} \sum_{i=1}^{5} e_i =$

$b_1 = b_1 - \alpha * \frac{1}{5} \sum_{i=1}^{5} e_i * x_i =$

Iteration 2:

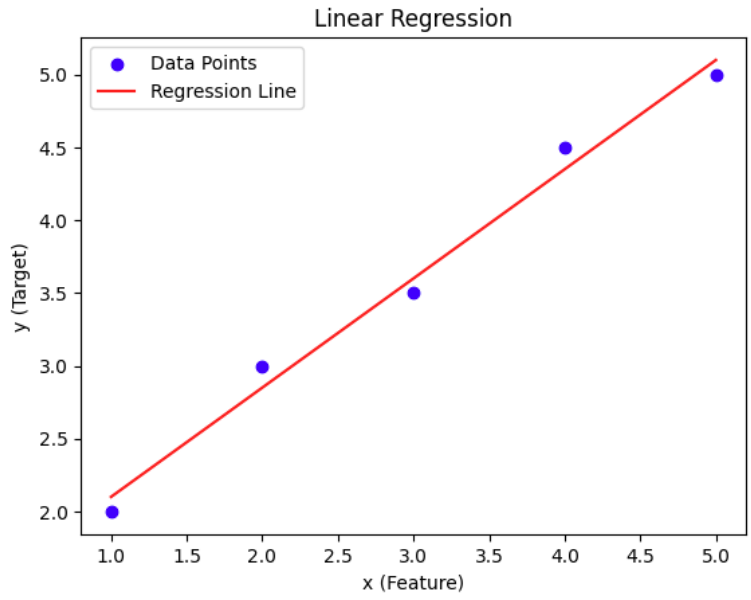
x (feature)	y (target)	$\hat{y}$ (predicted)	$e = \hat{y} - y$
3	3.5		
1	2		
2	3		
4	4.5		
5	5		

Iteration 3:

x (feature)	y (target)	$\hat{y}$ (predicted)	$e = \hat{y} - y$
3	3.5		
1	2		
2	3		
4	4.5		
5	5		

1.3 Linear Regression with Sci-Kit Learn

```python
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Define the dataset
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)  # Reshape to a 2D array
Y = np.array([2, 3, 3.5, 4.5, 5])

# Create a linear regression model
model = LinearRegression()

# Fit the model to the data
model.fit(X, Y)

# Get the coefficients (slope and intercept)
slope = model.coef_[0]
intercept = model.intercept_

# Make predictions for the same X values
Y_train_pred = model.predict(X)

# Plot the data points and the regression line
plt.scatter(X, Y, label='Data Points', color='blue')
plt.plot(X, Y_train_pred, label='Regression Line', color='red')
plt.xlabel('x (Feature)')
plt.ylabel('y (Target)')
plt.legend()
plt.title('Linear Regression')
plt.show()

# Print the coefficients
print(f"Slope (b1): {slope}")
print(f"Intercept (b0): {intercept}")
```



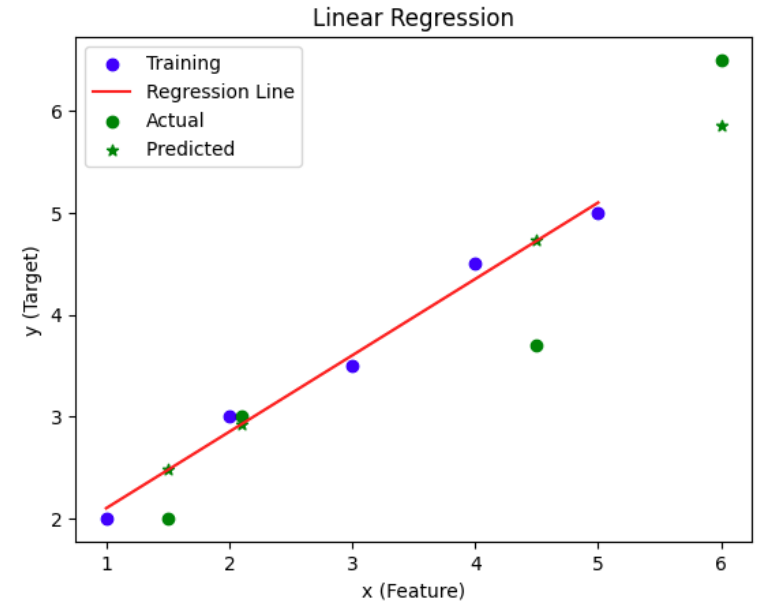

```
Slope (b1): 0.75
Intercept (b0): 1.35
```

Practice:
Predict the testing data using the created Linear Regression Model

$$\hat{y} = b_0 + b_1 x$$

where $b_0 = 1.35$ and $b_1 = 0.75$

X_test (Feature)	Y_test (Actual)	Y_predicted
1.5	2.0	
2.1	3.0	
4.5	3.7	
6.0	6.5	



1.4 Evaluation Metrics for Regression

In regression tasks, several evaluation metrics can be used to assess the performance of a model. Here are some common evaluation metrics:

- **Mean Squared Error (MSE)**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Root Mean Squared Error (RMSE)**:

$$\text{RMSE} = \sqrt{\text{MSE}}$$

- **Mean Absolute Error (MAE)**:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **R-squared (R^2)**:

$$R^2 = 1 - \frac{SSR}{SST}$$

where SSR is the sum of squares of residuals, and SST is the total sum of squares.

$$SSR = \sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

$$SST = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

n is the number of data points.
$\hat{y}$ is the predicted value for the i-th data point.
y_i is the actual target value for the i-th data point.
$\bar{y}$ is the mean of all the actual target values.

Practice:
We have a dataset with actual target values (Y) and predicted values ($\hat{Y}$) as follows:

x (Feature)	y (Actual)	$\hat{y}$ (Predicted)
1.5	2.0	2.475
2.1	3.0	2.925
4.5	3.7	4.725
6.0	6.5	5.85

$MSE =$

$RMSE =$

$MAE =$

$SSR =$

$SST =$

$R^2 =$

Evaluation Metrics for Regression with Python

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error,
                                        r2_score

# Define the actual and predicted values
actual_values = np.array([2.0, 3.0, 3.7, 6.5])
predicted_values = np.array([2.475, 2.925, 4.725, 5.85])

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(actual_values, predicted_values)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Calculate R-squared (R^2)
r2 = r2_score(actual_values, predicted_values)

# Print the results
print(f"Mean Squared Error (MSE): {np.round(mse,2)}")
print(f"Root Mean Squared Error (RMSE): {np.round(rmse,2)}")
print(f"R-squared (R^2): {np.round(r2,2)}")
```

```
Mean Squared Error (MSE): 0.43
Root Mean Squared Error (RMSE): 0.65
R-squared (R^2): 0.85
```

1.5 LinearRegression Arguments in Sci-Kit Learn

The ‘LinearRegression‘ class in scikit-learn doesn’t take any positional arguments, but it accepts several keyword arguments (arguments passed by name) when creating an instance of the model. Here are the commonly used keyword arguments (arguments and their default values):

```
LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs
                                    =None)
```

- fit_intercept (boolean, default=True):
  Whether to calculate the intercept (bias) for the model. If set to ‘False‘, the model will be forced to pass through the origin (0, 0).
- normalize (boolean, default=False):
  If ‘True‘, the input features (X) will be normalized before fitting the model. This parameter is ignored when ‘fit_intercept‘ is set to ‘False‘.
- copy_X (boolean, default=True):
  If ‘True‘, a copy of the input features (X) will be made before fitting the model. Setting it to ‘False‘ can save memory if you don’t need to keep the original data.
- n_jobs (int, default=None):
  The number of CPU cores to use when calculating the coefficients. If set to ‘None‘, it uses a single core. Setting it to -1 uses all available cores.

You create an instance of ‘LinearRegression‘ and pass these keyword arguments as needed. For example:

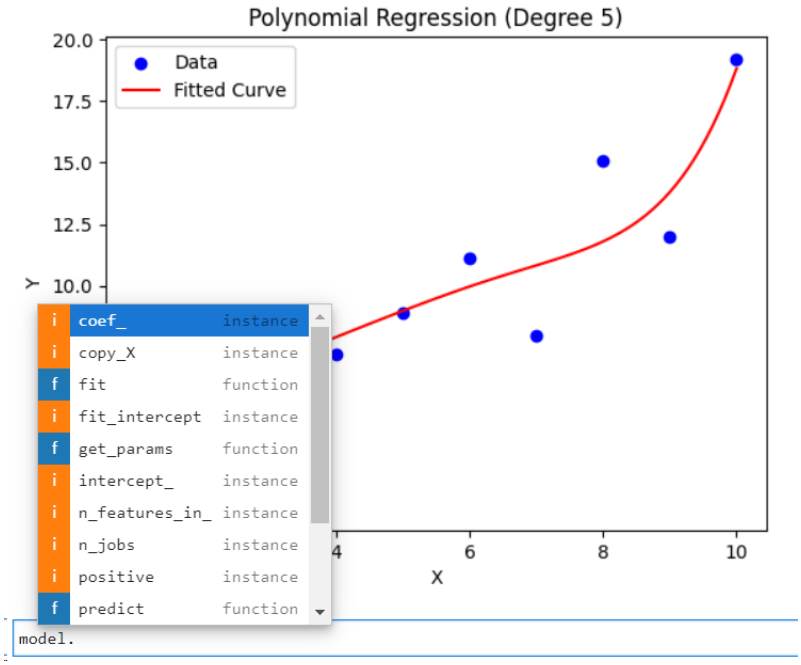
```
from sklearn.linear_model import LinearRegression

# Create a LinearRegression model with custom arguments
model = LinearRegression(fit_intercept=True, normalize=False, copy_X=True
                                    , n_jobs=None)
```

These keyword arguments allow you to configure the behavior of the linear regression model based on your specific requirements and the characteristics of your dataset.

1.6 Nonlinear Regression

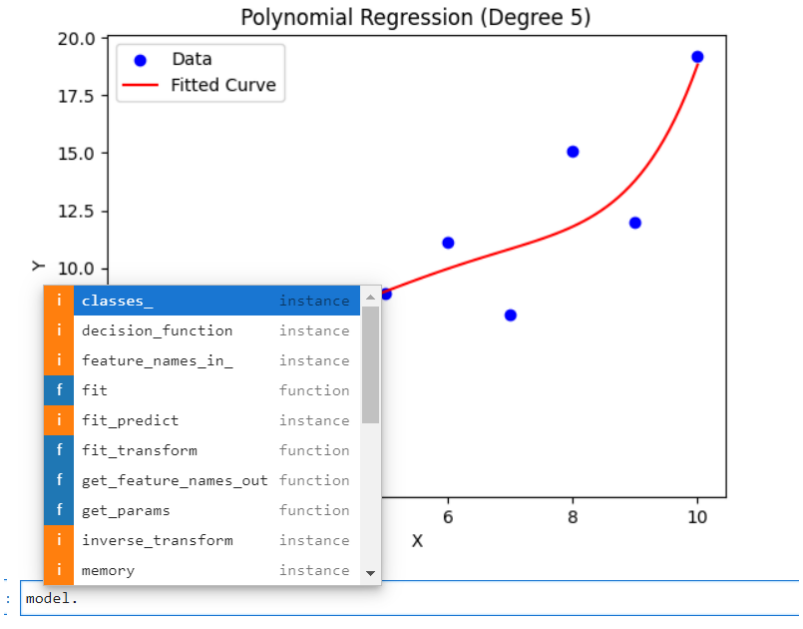
Nonlinear Regression Without Pipeline:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Define the given dataset
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
Y = np.array([1.2, 4, 8, 7.2, 8.9, 11.1, 8, 15.1, 12.0, 19.2])

# Create polynomial features (e.g., quadratic)
degree = 5
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)

# Create and fit a linear regression model
model = LinearRegression()
model.fit(X_poly, Y)

# Generate predictions
X_test = np.linspace(1, 10, 100).reshape(-1, 1)
X_test_poly = poly_features.transform(X_test)
Y_pred = model.predict(X_test_poly)

# Plot the original data and the fitted curve
plt.scatter(X, Y, label='Data', color='blue')
plt.plot(X_test, Y_pred, label='Fitted Curve', color='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title(f'Polynomial Regression (Degree {degree})')
plt.show()
```



Nonlinear Regression With Pipeline:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

# Define the given dataset
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
Y = np.array([1.2, 4, 8, 7.2, 8.9, 11.1, 8, 15.1, 12.0, 19.2])

# Create a pipeline for polynomial regression
degree = 5
model = Pipeline([
('poly_features', PolynomialFeatures(degree=degree)),
('linear_regression', LinearRegression())
])

# Fit the model to the data
model.fit(X, Y)

# Generate predictions
X_test = np.linspace(1, 10, 100).reshape(-1, 1)
Y_pred = model.predict(X_test)

# Plot the original data and the fitted curve
plt.scatter(X, Y, label='Data', color='blue')
plt.plot(X_test, Y_pred, label='Fitted Curve', color='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title(f'Polynomial Regression (Degree {degree})')
plt.show()
```



1.7 Save and Load Model

Save Model

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
import joblib

# Define the given dataset
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
Y = np.array([1.2, 4, 8, 7.2, 8.9, 11.1, 8, 15.1, 12.0, 19.2])

# Create a pipeline for polynomial regression
degree = 5
model = Pipeline([
('poly_features', PolynomialFeatures(degree=degree)),
('linear_regression', LinearRegression())
])

# Fit the model to the data
model.fit(X, Y)

# Save the model to a file
model_filename = 'polynomial_regression_model.pkl'
joblib.dump(model, model_filename)
```

Load Model

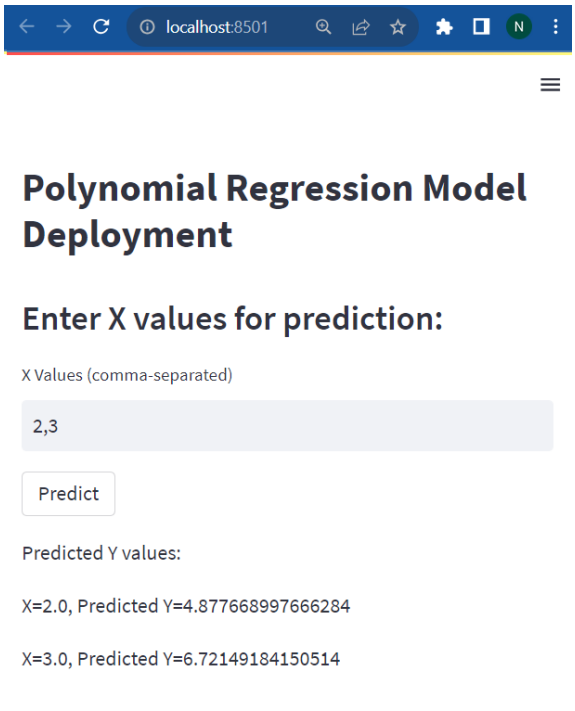
```
import numpy as np
import joblib

# Load the saved model
model_filename = 'polynomial_regression_model.pkl'
loaded_model = joblib.load(model_filename)

# Generate predictions using the loaded model
X_test = np.array([6, 2.5, 3.4, 4, 5.2]).reshape(-1,1)
Y_pred = loaded_model.predict(X_test)
print(Y_pred)
```

1.8 Deploy Model with Streamlit on Local

File: appxxx.py

```
import streamlit as st
import numpy as np
import joblib

# Load the saved model
model_filename = 'polynomial_regression_model.pkl'
loaded_model = joblib.load(model_filename)

# Streamlit UI
st.title('Polynomial Regression Model Deployment')

# User input for X values
st.header('Enter X values for prediction:')
x_input = st.text_input('X Values (comma-separated)', '6,2.5,3.4,4,5.2')

# Convert user input to a NumPy array
try:
x_input = np.array([float(x.strip()) for x in x_input.split(',')]).reshape(
                                        -1, 1)
except ValueError:
st.error('Invalid input. Please enter comma-separated numeric values.')
x_input = None

# Predict and display results when the "Predict" button is clicked
if st.button('Predict'):
if x_input is not None:
st.write('Predicted Y values:')
y_pred = loaded_model.predict(x_input)
for i, pred in enumerate(y_pred):
st.write(f'X={x_input[i][0]}, Predicted Y={pred}')
```

Run in Terminal: streamlit run appxx.py

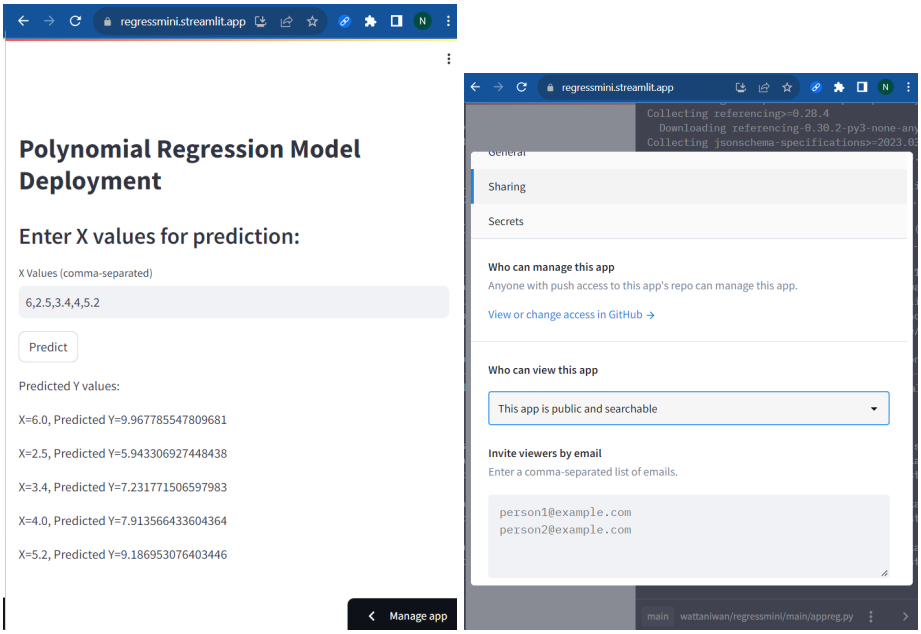


1.9 Deploy Model with Streamlit on Streamlit Sharing

1. **Prepare Your Streamlit App**: Make sure you have a Streamlit app script ready. This script should contain the code for your application, including any necessary data loading, model loading, and UI design.
2. **Create a Requirements File**: Create a ‘requirements.txt‘ file that lists all the Python packages your app depends on. This file is necessary for Streamlit Sharing to install the required packages.
   For example, your ‘requirements.txt‘ might look like this:

   ```
   streamlit
   numpy
   scikit-learn
   joblib
   ```

3. **Set Up a GitHub Repository:** Push your Streamlit app code, model file and the ‘requirements.txt‘ file to this repository.
4. **Create a Streamlit Sharing Account:** If you don’t have one, sign up for a Streamlit Sharing account at https://share.streamlit.io/.
5. **Connect GitHub to Streamlit Sharing:** After creating an account, log in to Streamlit Sharing and connect your GitHub account to it. This allows Streamlit Sharing to access your GitHub repository.
6. **Create a New App on Streamlit Sharing:** Click on "New App" in Streamlit Sharing and select your GitHub repository. Follow the instructions to set up your app.
7. **Specify Deployment Settings:** Configure the deployment settings for your app.
8. **Deploy Your Streamlit App:** Click the "Deploy" button in Streamlit Sharing. The service will build and deploy your app.
9. **View Your Deployed App:** Once the deployment process is complete, you will be provided with a URL where your Streamlit app is hosted. You can share this URL with others to access your app.
10. **Manage and Update Your App:** - You can update your app by pushing changes to your GitHub repository. Streamlit Sharing will automatically rebuild and redeploy your app when it detects changes in your repository.



1.10 Assignment

- Choose a real-world dataset that contains at **least three features** and a numeric target variable that you want to predict. Ensure the dataset is suitable for regression analysis.
- Create a regression model to predict a target variable based on features from the dataset.
- Deploy the regression model as a web application using Streamlit.io.
- Put the evaluation result on your web app.
- In the Streamlit Sharing settings for your app, make sure to set it to "Public" or "Anyone with the link can view."