# **Breakout Sessions Handout**

## **Breakout Session 1:**

|  |
| --- |
| *"""*  Introduction to Polynomial and Interaction Features  --------------------------------------------------  STUDENT TASKS:  1. Complete the functions to create polynomial features (squares and cubes)  2. Complete the interaction features function to create pairwise combinations  3. Visualize how these new features affect the ability to separate classes  4. Determine which derived features have the strongest correlation with the target  5. Create one domain-specific polynomial or interaction feature of your own design  This starter code provides the basic structure for exploring how transforming  features can reveal non-linear relationships and improve model performance.  """  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.datasets import make\_moons  from sklearn.preprocessing import PolynomialFeatures  # Generate a synthetic non-linear dataset (moons dataset)  np.random.seed(42)  X, y = make\_moons(n\_samples=1000, noise=0.2, random\_state=42)  # Create a dataframe for easier manipulation  df = pd.DataFrame(X, columns=['feature1', 'feature2'])  df['target'] = y  print("Original Dataset:")  print(df.head())  print("\nSummary Statistics:")  print(df.describe())  # TODO: Function to create polynomial features (square and cubic terms)  def create\_polynomial\_features(dataframe):  """  Create polynomial features (squared and cubed) for each numeric feature.  Args:  dataframe: The input pandas DataFrame  Returns:  DataFrame with added polynomial features  """  df\_poly = dataframe.copy()  # Your code here:  # 1. Create squared features (feature1^2, feature2^2)  # 2. Create cubic features (feature1^3, feature2^3)  return df\_poly  # TODO: Function to create interaction features  def create\_interaction\_features(dataframe):  """  Create interaction features (multiplication of feature pairs).  Args:  dataframe: The input pandas DataFrame  Returns:  DataFrame with added interaction features  """  df\_interact = dataframe.copy()  # Your code here:  # Create feature interactions (e.g., feature1 \* feature2)  return df\_interact  # Combine all features  def enhance\_features(dataframe):  """  Apply both polynomial and interaction transformations.  Args:  dataframe: The input pandas DataFrame  Returns:  DataFrame with all enhanced features  """  # First add polynomial features  df\_enhanced = create\_polynomial\_features(dataframe)  # Then add interaction features  df\_enhanced = create\_interaction\_features(df\_enhanced)  return df\_enhanced  # Apply feature enhancement  enhanced\_df = enhance\_features(df)  # TODO: Complete this visualization to compare original vs. enhanced features  def visualize\_features(original\_df, enhanced\_df):  """  Create visualizations comparing original and enhanced features.  """  fig, axes = plt.subplots(2, 2, figsize=(14, 10))  # Original features visualization  axes[0, 0].scatter(original\_df['feature1'], original\_df['feature2'],  c=original\_df['target'], cmap='viridis', alpha=0.6)  axes[0, 0].set\_title('Original Features: feature1 vs feature2')  axes[0, 0].set\_xlabel('feature1')  axes[0, 0].set\_ylabel('feature2')  # TODO: Add three more visualizations showing your enhanced features  # Suggestion: Show interaction features, polynomial features, and a combination  # Example (uncomment and modify):  # axes[0, 1].scatter(enhanced\_df['feature1\_squared'], enhanced\_df['feature2'],  # c=enhanced\_df['target'], cmap='viridis', alpha=0.6)  # axes[0, 1].set\_title('Enhanced Features: feature1\_squared vs feature2')  # axes[0, 1].set\_xlabel('feature1\_squared')  # axes[0, 1].set\_ylabel('feature2')  plt.tight\_layout()  plt.savefig('polynomial\_features\_visualization.png')  plt.show()  # Visualization of features  visualize\_features(df, enhanced\_df)  # TODO: Analyze the correlation of features with the target  def analyze\_feature\_importance(enhanced\_df):  """  Calculate and display correlation of features with target.  """  # Your code here:  # 1. Calculate correlation of all features with target  # 2. Sort correlations in descending order  # 3. Display the top features  pass  # Feature importance analysis  analyze\_feature\_importance(enhanced\_df)  # TODO: Create one domain-specific feature of your own design  def create\_custom\_feature(dataframe):  """  Create a custom feature that might be relevant for this dataset.  Args:  dataframe: The input pandas DataFrame  Returns:  DataFrame with added custom feature  """  df\_custom = dataframe.copy()  # Your code here:  # Create a custom feature that you think might be useful  # For example: distance from origin, angle, or another transformation  return df\_custom  # Apply your custom feature  final\_df = create\_custom\_feature(enhanced\_df)  # Save the enhanced dataset  final\_df.to\_csv('dataset\_with\_polynomial\_features.csv', index=False)  print("\nEnhanced dataset saved as 'dataset\_with\_polynomial\_features.csv'")  # BONUS: Try using scikit-learn's PolynomialFeatures  # Compare your manual implementation with scikit-learn's implementation  def compare\_with\_sklearn(original\_df):  """  Compare manual polynomial features with scikit-learn's implementation.  """  # Extract features (exclude target)  X = original\_df.drop('target', axis=1).values  # Create polynomial features using scikit-learn  poly = PolynomialFeatures(degree=3, include\_bias=False)  X\_poly = poly.fit\_transform(X)  # Create DataFrame with sklearn's polynomial features  feature\_names = poly.get\_feature\_names\_out(['feature1', 'feature2'])  df\_sklearn\_poly = pd.DataFrame(X\_poly, columns=feature\_names)  df\_sklearn\_poly['target'] = original\_df['target'].values  print("\nScikit-learn PolynomialFeatures Output:")  print(df\_sklearn\_poly.head())  return df\_sklearn\_poly  # Uncomment to compare with scikit-learn  # sklearn\_poly\_df = compare\_with\_sklearn(df) |

## **Breakout Session 2:**

|  |
| --- |
| *"""*  Introduction to Polynomial and Interaction Features  --------------------------------------------------  STUDENT TASKS:  1. Complete the functions to create polynomial features (squares and cubes)  2. Complete the interaction features function to create pairwise combinations  3. Visualize how these new features affect the ability to separate classes  4. Determine which derived features have the strongest correlation with the target  5. Create one domain-specific polynomial or interaction feature of your own design  This starter code provides the basic structure for exploring how transforming  features can reveal non-linear relationships and improve model performance.  """  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.datasets import make\_moons  from sklearn.preprocessing import PolynomialFeatures  # Generate a synthetic non-linear dataset (moons dataset)  np.random.seed(42)  X, y = make\_moons(n\_samples=1000, noise=0.2, random\_state=42)  # Create a dataframe for easier manipulation  df = pd.DataFrame(X, columns=['feature1', 'feature2'])  df['target'] = y  print("Original Dataset:")  print(df.head())  print("\nSummary Statistics:")  print(df.describe())  # TODO: Function to create polynomial features (square and cubic terms)  def create\_polynomial\_features(dataframe):  """  Create polynomial features (squared and cubed) for each numeric feature.  Args:  dataframe: The input pandas DataFrame  Returns:  DataFrame with added polynomial features  """  df\_poly = dataframe.copy()  # Your code here:  # 1. Create squared features (feature1^2, feature2^2)  # 2. Create cubic features (feature1^3, feature2^3)  return df\_poly  # TODO: Function to create interaction features  def create\_interaction\_features(dataframe):  """  Create interaction features (multiplication of feature pairs).  Args:  dataframe: The input pandas DataFrame  Returns:  DataFrame with added interaction features  """  df\_interact = dataframe.copy()  # Your code here:  # Create feature interactions (e.g., feature1 \* feature2)  return df\_interact  # Combine all features  def enhance\_features(dataframe):  """  Apply both polynomial and interaction transformations.  Args:  dataframe: The input pandas DataFrame  Returns:  DataFrame with all enhanced features  """  # First add polynomial features  df\_enhanced = create\_polynomial\_features(dataframe)  # Then add interaction features  df\_enhanced = create\_interaction\_features(df\_enhanced)  return df\_enhanced  # Apply feature enhancement  enhanced\_df = enhance\_features(df)  # TODO: Complete this visualization to compare original vs. enhanced features  def visualize\_features(original\_df, enhanced\_df):  """  Create visualizations comparing original and enhanced features.  """  fig, axes = plt.subplots(2, 2, figsize=(14, 10))  # Original features visualization  axes[0, 0].scatter(original\_df['feature1'], original\_df['feature2'],  c=original\_df['target'], cmap='viridis', alpha=0.6)  axes[0, 0].set\_title('Original Features: feature1 vs feature2')  axes[0, 0].set\_xlabel('feature1')  axes[0, 0].set\_ylabel('feature2')  # TODO: Add three more visualizations showing your enhanced features  # Suggestion: Show interaction features, polynomial features, and a combination  # Example (uncomment and modify):  # axes[0, 1].scatter(enhanced\_df['feature1\_squared'], enhanced\_df['feature2'],  # c=enhanced\_df['target'], cmap='viridis', alpha=0.6)  # axes[0, 1].set\_title('Enhanced Features: feature1\_squared vs feature2')  # axes[0, 1].set\_xlabel('feature1\_squared')  # axes[0, 1].set\_ylabel('feature2')  plt.tight\_layout()  plt.savefig('polynomial\_features\_visualization.png')  plt.show()  # Visualization of features  visualize\_features(df, enhanced\_df)  # TODO: Analyze the correlation of features with the target  def analyze\_feature\_importance(enhanced\_df):  """  Calculate and display correlation of features with target.  """  # Your code here:  # 1. Calculate correlation of all features with target  # 2. Sort correlations in descending order  # 3. Display the top features  pass  # Feature importance analysis  analyze\_feature\_importance(enhanced\_df)  # TODO: Create one domain-specific feature of your own design  def create\_custom\_feature(dataframe):  """  Create a custom feature that might be relevant for this dataset.  Args:  dataframe: The input pandas DataFrame  Returns:  DataFrame with added custom feature  """  df\_custom = dataframe.copy()  # Your code here:  # Create a custom feature that you think might be useful  # For example: distance from origin, angle, or another transformation  return df\_custom  # Apply your custom feature  final\_df = create\_custom\_feature(enhanced\_df)  # Save the enhanced dataset  final\_df.to\_csv('dataset\_with\_polynomial\_features.csv', index=False)  print("\nEnhanced dataset saved as 'dataset\_with\_polynomial\_features.csv'")  # BONUS: Try using scikit-learn's PolynomialFeatures  # Compare your manual implementation with scikit-learn's implementation  def compare\_with\_sklearn(original\_df):  """  Compare manual polynomial features with scikit-learn's implementation.  """  # Extract features (exclude target)  X = original\_df.drop('target', axis=1).values  # Create polynomial features using scikit-learn  poly = PolynomialFeatures(degree=3, include\_bias=False)  X\_poly = poly.fit\_transform(X)  # Create DataFrame with sklearn's polynomial features  feature\_names = poly.get\_feature\_names\_out(['feature1', 'feature2'])  df\_sklearn\_poly = pd.DataFrame(X\_poly, columns=feature\_names)  df\_sklearn\_poly['target'] = original\_df['target'].values  print("\nScikit-learn PolynomialFeatures Output:")  print(df\_sklearn\_poly.head())  return df\_sklearn\_poly  # Uncomment to compare with scikit-learn  # sklearn\_poly\_df = compare\_with\_sklearn(df) |