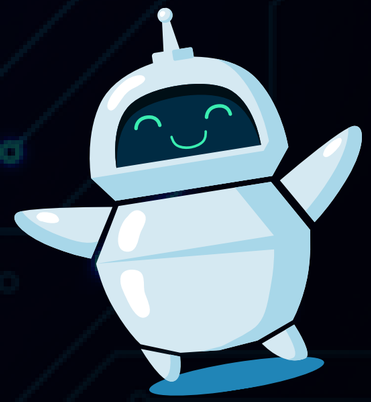
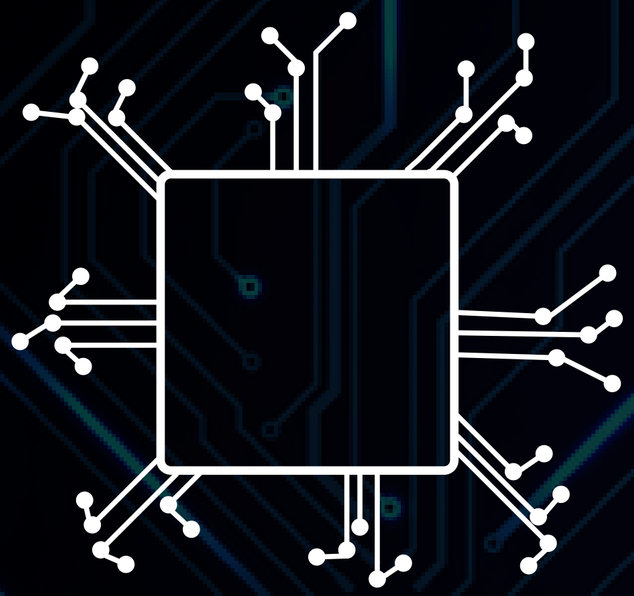
Machine Learning

Non-Linear and Logistic Regression



INTRODUCTION

This project has two main parts:
Non-Linear Regression and Logistic Regression.
The goal is to understand how model complexity and regularization affect performance and generalization.



Part 1: Non-linear Regression

- Data Generation
- Ridge Regression
- RBF Regression

Data Generation

- We generated a synthetic dataset using the function: $y=\sin(5\pi x)+\epsilon i$.
- The dataset contains 25 points with small random noise.

```
# Generate dataset S
np.random.seed(25) # makes the random numbers repeatable

n = 25 # number of data points
x = np.random.uniform(0, 1, n) # generates 25 random x-values between 0 and 1
noise = np.random.uniform(-0.3, 0.3, n) # generate random noise for each point, between -0.3 and +0.3
y = np.sin(5 * np.pi * x) + noise # generates the y values using the formula= sin(5πx) + noise

xs = np.linspace(0, 1, 400) # makes smooth curves
ys_true = np.sin(5 * np.pi * xs) # computes the true function without noise
```

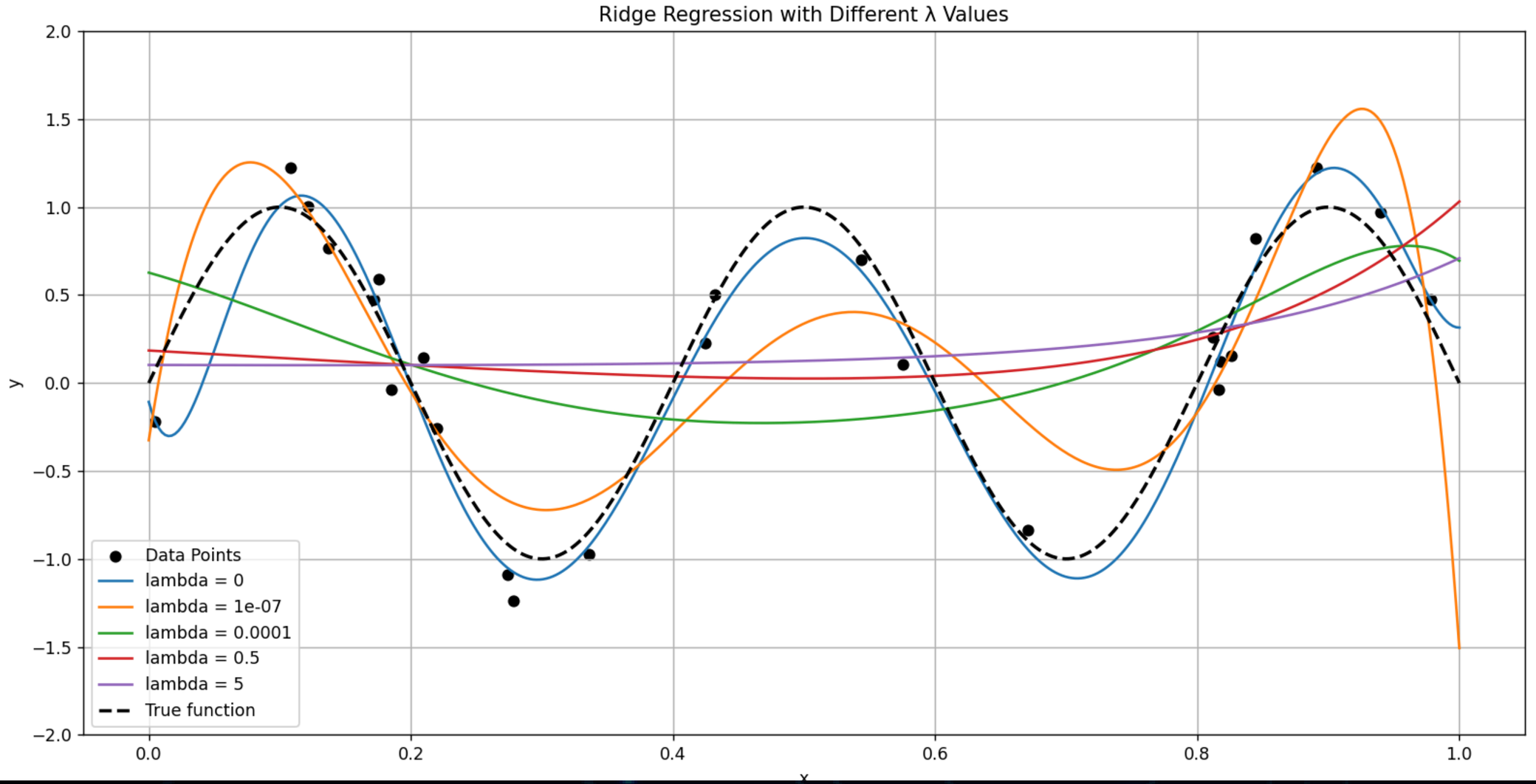
Ridge Regression

- Ridge Regression is a method used to fit polynomial models while controlling model complexity.
- We applied Ridge Regression using five different values of λ.
- We observed how λ changes the smoothness of the curve:

```
# part A #
# Polynomial features (degree = 9)
degree = 9 # sets the polynomial degree = 9
poly = PolynomialFeatures(degree) # creates a PolynomialFeatures (This can make curves, waves, and smooth shapes, not just straight lines)object
X = poly.fit_transform(x.reshape(-1, 1)) # Converts x-values into polynomial form:[1, x, x², x³, …, x¹⁰]
Xs = poly.transform(xs.reshape(-1, 1))

# Ridge regression for different λ
lambdas = [0, 0.0000001, 0.0001, 0.5, 5] #List of λ (regularization strengths) to compare
plt.figure(figsize=(2, 7))
plt.scatter(x, y, color='black', label="Data Points") #Plots the original noisy data points

for lam in lambdas: #Loop over each λ value
    model = Ridge(alpha=lam) #Creates a Ridge model with regularization strength λ
    model.fit(X, y) #Trains the model using the polynomial features
    y_pred = model.predict(Xs) #Predicts values on the smooth xs grid
    plt.plot(xs, y_pred, label=f"lambda = {lam}")
```


Ridge Regression with Different λ Values
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
Data Points
lambda = 0
lambda = 1e-07
lambda = 0.0001
lambda = 0.5
lambda = 5
True function

- λ = 0 (No regularization) – Best fit
  This curve follows the true function most accurately.
  Captures all the waves and peaks of sin(5πx).
  Matches the data points very well.
  → Best performance because our data set is small.

- Moderate λ (0.5) – Underfitting begins
  The curve becomes too flat.
  The model loses the shape of sin(5πx).
  Doesn't follow the oscillation pattern anymore.
  → Beginning of underfitting.

- Very small λ (1e−7 and 0.0001) – Slightly smoother
  These values add a tiny amount of regularization.
  The curves are still flexible, but slightly less accurate than λ = 0.
  Some peaks and valleys are reduced.
  → Still good, but not as precise as λ = 0.

- Large λ (5) – Strong underfitting
  The model becomes almost a straight line.
  Completely ignores the data pattern.
  Cannot capture any wave structure.
  → Very poor fit.

Conclusion

- For our small dataset (25 points), λ = 0 gives the best generalization.
- The more we increase λ, the flatter the curve becomes, leading to underfitting.

RBF Regression

- We tested models with different numbers of RBF basis functions: 1, 5, 10, and 50.
- Key observations:
  1 RBF → Too simple (underfitting)
  50 RBF → Too complex (overfitting)
  Best generalization: 5 or 10 RBFs

- we chose sigma = 1/M because it Produced smoother and more accurate approximations of the true function. and it had less extreme oscillations and followed the sin(5πx) shape more closely

```
# part B #
# RBF function
# Computes a Gaussian "bump" centered at c with width sigma
# x: input points c: center of the RBF sigma: controls the width (spread) of the RBF
def rbf(x, c, sigma):
    return np.exp(-(x - c)**2 / (2 * sigma**2))

rbf_counts = [1, 5, 10, 50] # Different numbers of RBFs

plt.figure(figsize=(12, 10))

for i, M in enumerate(rbf_counts, 1):
    centers = np.linspace(0, 1, M)
    # Choose RBF width (sigma): smaller when we have more RBFs
    sigma = 1 / (M)  # reasonable width

    # Construct RBF feature matrix
    Phi = np.vstack([rbf(x, c, sigma) for c in centers]).T
    Phi_s = np.vstack([rbf(xs, c, sigma) for c in centers]).T

    # Linear regression
    model = LinearRegression()
    model.fit(Phi, y)
    y_pred = model.predict(Phi_s)
```

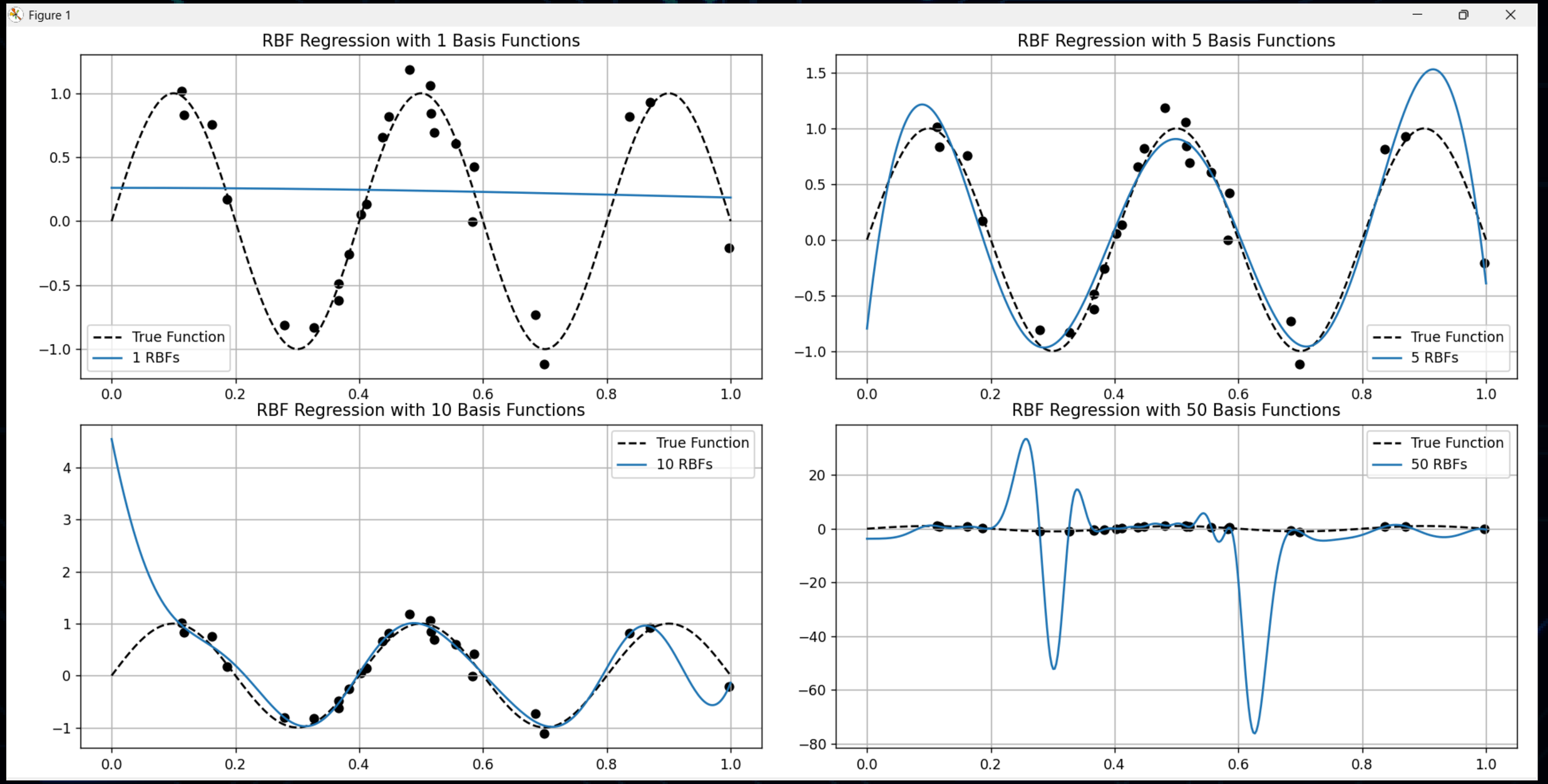


Figure 1
RBF Regression with 1 Basis Functions
True Function
1 RBFs
RBF Regression with 5 Basis Functions
True Function
5 RBFs
RBF Regression with 10 Basis Functions
True Function
10 RBFs
RBF Regression with 50 Basis Functions
True Function
50 RBFs

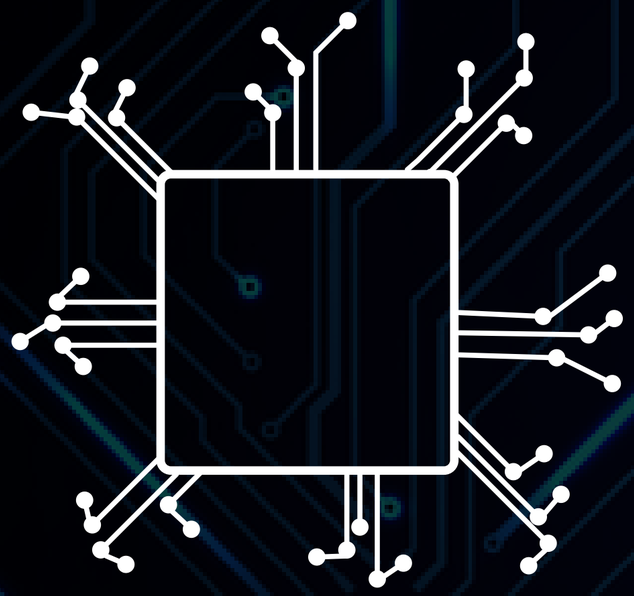
- 1 RBF
  Too simple
  Almost a straight line
  Doesn’t follow the wave
  ➡ Underfitting

- 10 RBFs
  More flexible
  Starts to bend too much at the edges
  ➡ Slight overfitting

- 5 RBFs
  Smooth curve
  Closest to the true sin(5πx) function
  Good balance between simple and complex
  ➡ Best generalization

- 50 RBFs
  Extremely wiggly
  Follows noise instead of the true function
  ➡ Strong overfitting

Conclusion:
The model with 5 RBFs gives the best generalization because it follows the true function well without overfitting or underfitting.



Part 2: Logistic Regression

- Data Preprocessing
- Data Scaling and Splitting
- Linear Logistic Regression degree 1
- Linear Logistic Regression degree 1 result
- Nonlinear Logistic Regression degree 2,5,9
- Nonlinear Logistic Regression degree 2,5,9 result
- The ROC curve for the best model degree 5
- Handling Class Imbalance with Class Weighting

Data Preprocessing

- Loaded 3,500 customer records from Assignment 1.
- Dropped unnecessary columns (e.g., CustomerID).
- Handled missing values using column means.
- One-hot encoded categorical features.
- Standardized the data using StandardScaler.
- Split dataset into:
- ‣ 2500 training, 500 validation, 500 test samples

```
# Load dataset
csv_file = "C:/Users/Asus/OneDrive/Documents/ML_Ass2/customer_data.csv"
TARGET_COLUMN = "ChurnStatus"

data = pd.read_csv(csv_file)

# handle missing values
# replace missing values in numeric columns with the column mean:
data = data.fillna(data.mean(numeric_only=True))

y_raw = data[TARGET_COLUMN] #Extracts the ChurnStatus column from the DataFrame and stores it as y_raw

# Drop ChurnStatus + ID column from features
X = data.drop(columns=[TARGET_COLUMN, "CustomerID"])

# One-hot encode categorical columns
X = pd.get_dummies(X, drop_first=True)
```

Data Scaling and Splitting

- Split dataset into:
- 2500 training, 500 validation, 500 test samples
- We scale the data because logistic regression is sensitive to feature magnitude. Scaling ensures all features contribute equally, improves optimization, prevents instability, and is necessary when using polynomial features.

```
# Train/val/test split
total_samples = len(X) #total_samples is how many rows samples we have

# 2500 / 500 / 500
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, train_size=2500, random_state=0, stratify=y
) #First split so that train has exactly 2500 samples (train_size=2500) and the rest go into X_temp, y_temp.
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=0, stratify=y_temp
) #Then split X_temp, y_temp into 50% validation and 50% test.

print("Train:", len(X_train), "Val:", len(X_val), "Test:", len(X_test))

# Scaling --> Logistic Regression is sensitive to feature scale
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

Linear Logistic Regression degree 1

A linear decision boundary means logistic regression tries to separate churn vs. non-churn using a straight line, which is simple but often underfits complex data.
→ too simple → misses churners.

```
# Linear logistic regression (degree 1)
linear_model = LogisticRegression(max_iter=5000) #Creates a logistic regression model named linear_model.
linear_model.fit(X_train_scaled, y_train)

metrics_linear = evaluate_model_with_proba( #Calls the helper function to compute all metrics for this model
    linear_model,
    X_train_scaled, y_train,
    X_val_scaled, y_val,
    X_test_scaled, y_test
)
metrics_linear["degree"] = 1

print("\nLinear model metrics:")
print(metrics_linear)
```

Linear Logistic Regression degree 1 result

All models:

	train_acc	train_prec	train_rec	val_acc	val_prec	val_rec	test_acc	test_prec	test_rec	val_auc	test_auc	degree
0	0.9548	0.000000	0.000000	0.952	0.000000	0.000000	0.954	0.000000	0.000000	0.927445	0.883701	1

Results

- The linear model achieved high accuracy on all datasets (~95%).
- However, precision and recall are 0 for training, validation, and test sets.
- This happened because the model predicted all samples as “non-churn” (class 0).
- Since churners are the minority class, the model took the “easy” option and ignored them.
  → As a result, the model never predicted class 1, leading to:
  0 true positives, 0 false positives
- 100% of churners were missed

Interpretation

- The model underfits the data because the linear boundary is too simple.
- It cannot separate churners from non-churners using a straight line.
- The high accuracy is misleading because the dataset is imbalanced (mostly non-churn).
- The model performs poorly in identifying actual churners.

Conclusion

- A linear logistic regression model is not suitable for this dataset.
- We need a non-linear model (polynomial features) to capture more complex relationships.
- Although accuracy is high, the linear model completely fails to detect churners because its decision boundary is too simple.

Non_Linear Logistic Regression degree 2,5,9

Non-linear logistic regression uses polynomial features to create curved decision boundaries, allowing the model to capture more complex relationships in the data and separate churn vs. non-churn more effectively.

- Degree 2 draws a small curve → better.
- Degree 5 draws a smooth, flexible curve → best fit.
- Degree 9 draws a crazy wiggly shape → memorizes noise → overfits.

```
# Polynomial degrees 2, 5, 9
degrees = [2, 5, 9] #list of polynomial degrees
results = [metrics_linear]
models_info = [("degree=1", linear_model, None)]

for d in degrees:
    print(f"\nTraining polynomial degree {d} model...")
    poly = PolynomialFeatures(degree=d, include_bias=False)
    X_train_poly = poly.fit_transform(X_train_scaled) #Transforms X_train_scaled into polynomial features
    X_val_poly = poly.transform(X_val_scaled) #Transforms X_val_scaled into polynomial features
    X_test_poly = poly.transform(X_test_scaled) #Transforms X_test_scaled into polynomial features

    model = LogisticRegression(max_iter=5000)
    model.fit(X_train_poly, y_train)

    metrics_d = evaluate_model_with_proba(
        model,
        X_train_poly, y_train,
        X_val_poly, y_val,
        X_test_poly, y_test
    )
    metrics_d["degree"] = d
    results.append(metrics_d)
    models_info.append((f"degree={d}", model, poly))

results_df = pd.DataFrame(results)
print("\nAll models:")
print(results_df)
```

Non_Linear Logistic Regression degree 2,5,9 result

Degree 2

- More flexible than linear model
- Starts capturing real patterns in data
- Much better precision, recall, and AUC
- Generalizes well (good performance on val & test)

Degree 5 (Best Model)

- Even more flexible
- Fits the data well without memorizing noise
- Highest validation AUC (0.9949)
- Strong recall and precision
- Best generalization → performs well on unseen data
- Chosen as the best model

Degree 9

- Very complex model
- Extremely high training metrics → almost perfect
- Slight drop in validation/test metrics
- Indicates overfitting (memorizing noise)
- Worse generalization compared to degree 5

```
All models:
   train_acc  train_prec  train_rec  val_acc  val_prec   val_rec  test_acc  test_prec  test_rec   val_auc  test_auc  degree
0     0.9548    0.000000   0.000000    0.952  0.000000  0.000000     0.954   0.000000  0.000000  0.927445  0.883701       1
1     0.9604    0.882353   0.133929    0.952  0.400000  0.086957     0.954   0.000000  0.000000  0.956066  0.888646       2
2     0.9904    0.958333   0.821429    0.984  0.857143  0.782609     0.974   0.764706  0.590909  0.994896  0.865728       5
3     0.9972    1.000000   0.937500    0.986  0.807692  0.913043     0.972   0.700000  0.636364  0.994531  0.821558       9

Best model: degree=5
train_acc     0.990400
train_prec    0.958333
train_rec     0.821429
val_acc       0.984000
val_prec      0.857143
val_rec       0.782609
test_acc      0.974000
test_prec     0.764706
test_rec      0.590909
val_auc       0.994896
test_auc      0.865728
degree        5.000000
```

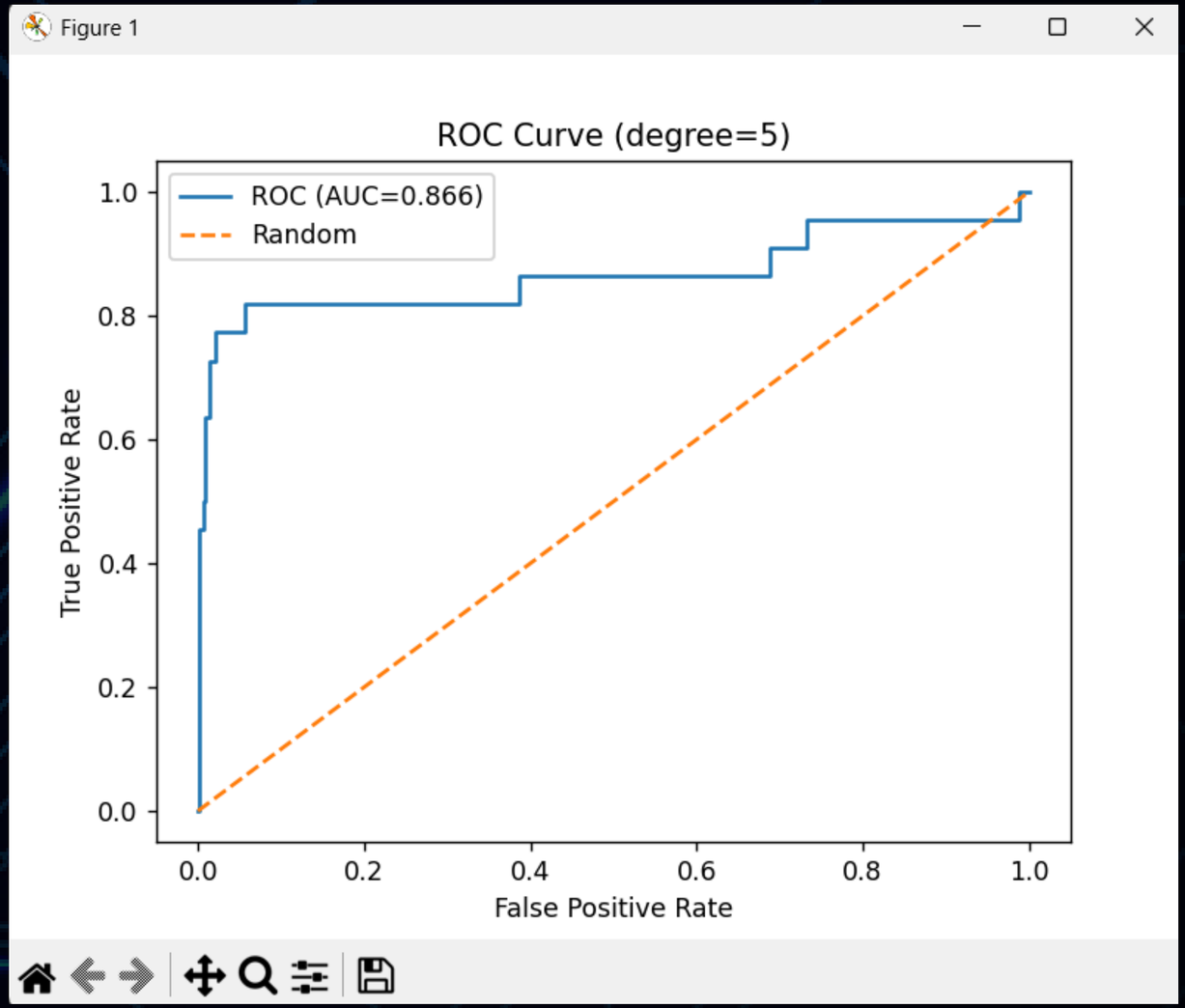
The ROC curve for the best model degree 5

AUC = 0.866, which means:

- Very good performance
- The model correctly ranks churn vs. non-churn in most cases
- Strong generalization to unseen test data

Summary

- Increasing degree → more flexible model
- Too low degree → underfitting
- Moderate degree (5) → best generalization
- Too high degree (9) → overfitting



Handling Class Imbalance with Class Weighting

- The dataset is imbalanced (few churners, many non-churn).
- Without weighting, the model ignored churners and predicted only class 0.
- We applied class_weight = "balanced" in logistic regression.
- This forces the model to give equal importance to churn and non-churn.
- As a result, precision and recall increased, and the model detected churners.

→ Class weighting fixes the imbalance and improves model performance.

Non_Linear Logistic Regression degree 2,5,9 result

- The model now correctly predicts churners (class 1).
- Precision and recall increased across all degrees.
- The model no longer predicts only “non-churn”.
- Class weighting fixed the imbalance problem.

Degree 1

- Big improvement compared to before (recall = 1.0!)
- But accuracy is still lower → still underfitting a bit.

Degree 2

- Better balance between precision and recall.
- Much stronger generalization than degree 1.

Degree 5 (Best Model)

- Highest validation accuracy (0.948)
- Very strong recall (1.0)
- Good test recall (0.777)
- Best validation AUC (0.991888)
- Best balance between learning and generalization

Degree 9

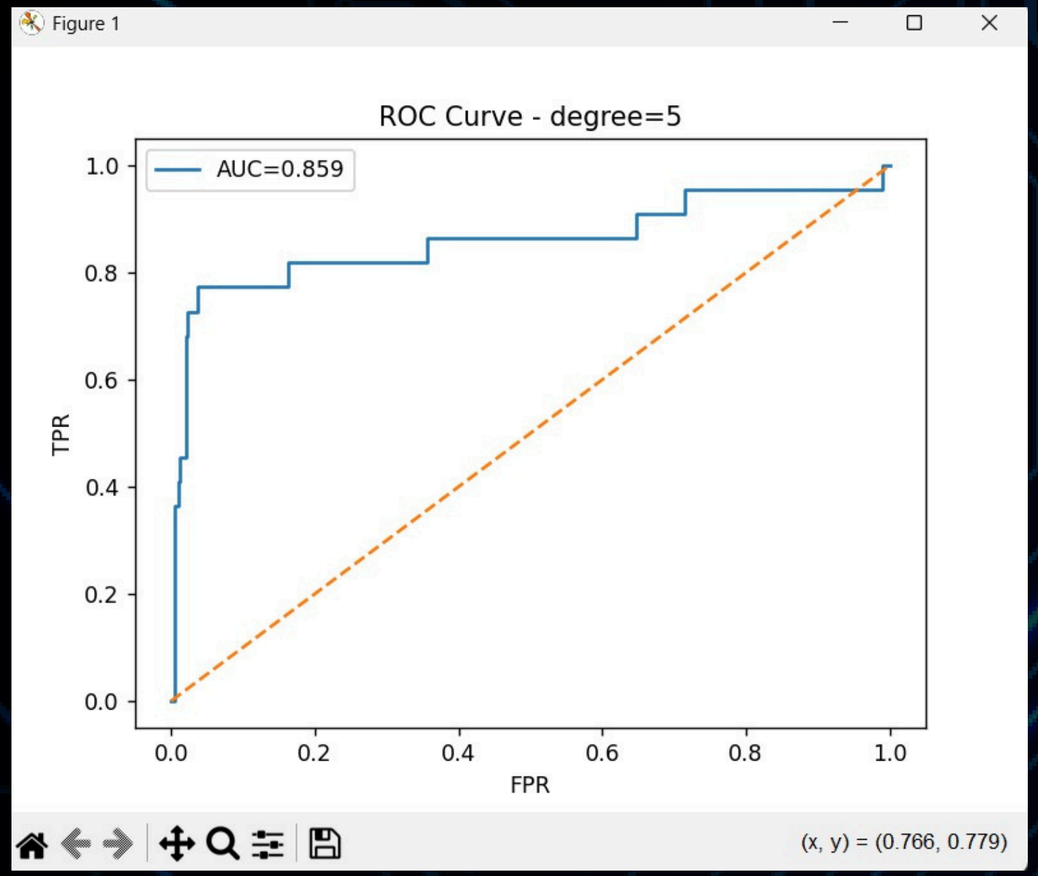
- Extremely high training recall (0.982) → signs of overfitting
- Validation and test performance slightly lower than degree 5
- Still good, but more complex than necessary

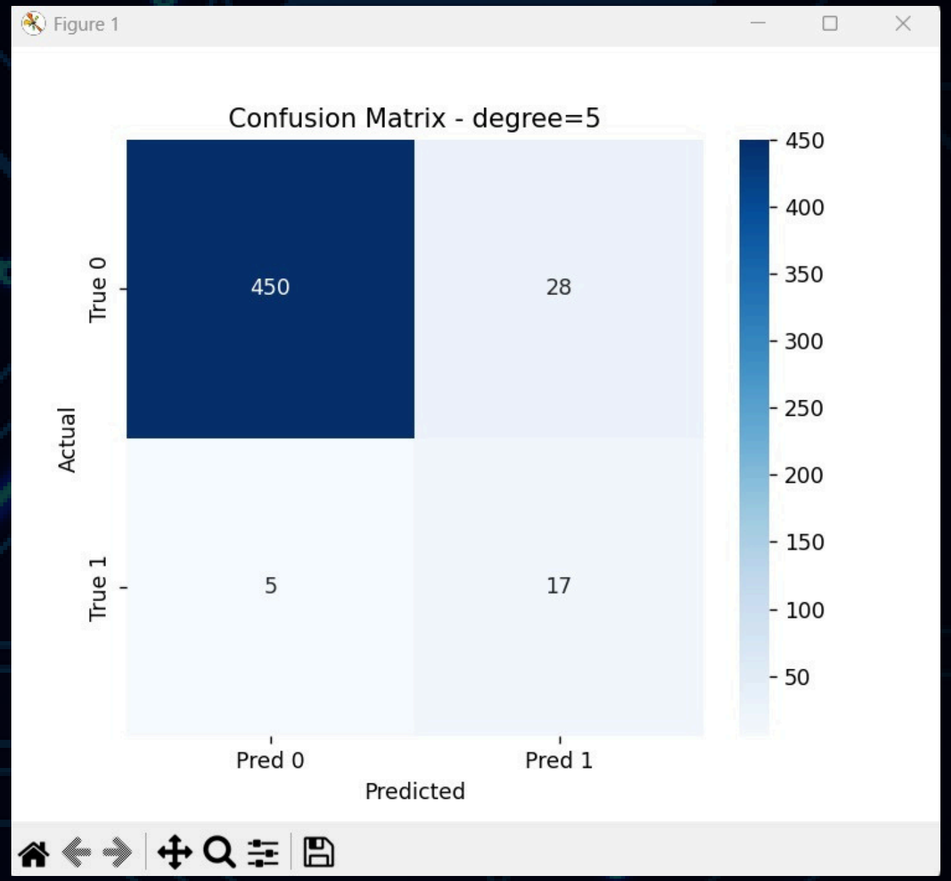
All models:

	train_acc	train_prec	train_rec	val_acc	val_prec	val_rec	test_acc	test_prec	test_rec	val_auc	test_auc	degree
0	0.8212	0.193784	0.946429	0.828	0.211009	1.000000	0.818	0.183486	0.909091	0.916781	0.875333	1
1	0.8768	0.259804	0.946429	0.882	0.280488	1.000000	0.870	0.227848	0.818182	0.965090	0.892355	2
2	0.9564	0.506977	0.973214	0.948	0.469388	1.000000	0.934	0.377778	0.772727	0.991888	0.859452	5
3	0.9644	0.558376	0.982143	0.950	0.478261	0.956522	0.928	0.340909	0.681818	0.987513	0.813950	9

Best model: degree=5

train_acc	0.956400
train_prec	0.506977
train_rec	0.973214
val_acc	0.948000
val_prec	0.469388
val_rec	1.000000
test_acc	0.934000
test_prec	0.377778
test_rec	0.772727
val_auc	0.991888
test_auc	0.859452
degree	5.000000

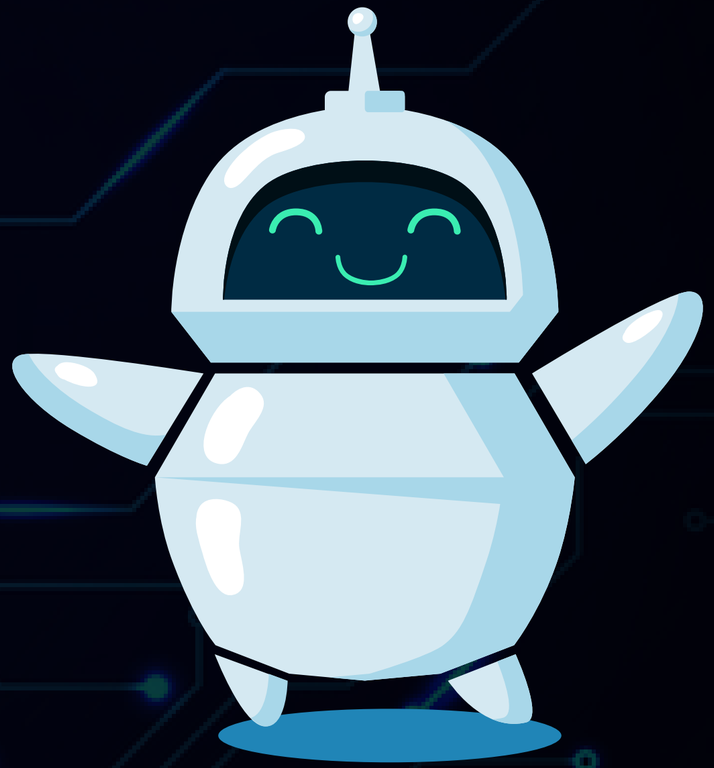




Class weighting helped the model finally detect class 1 (churn) and reduced the number of mistakes.
Degree-5 shows the best balance between catching churners and avoiding false alarms.

CONCLUSION

This assignment showed how model complexity and regularization affect learning. In Part 1, Ridge Regression and RBF models demonstrated that too little complexity causes underfitting, while too much leads to overfitting. The best results came from $\lambda = 0$ and using around 5 RBFs. In Part 2, linear logistic regression failed to detect churn due to its simple boundary, but polynomial models improved performance, with degree 5 achieving the best generalization. Class weighting further improved recall by addressing class imbalance. Overall, the best model was the degree-5 logistic regression with class weighting, providing strong and balanced performance.



prepared by:

Yasmin Al-Shawawrh 1220848
Basmala Abu-Hakema 1220184