Regularization and the kernel trick

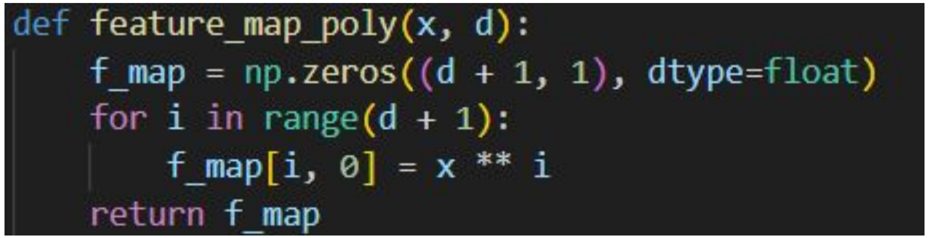
Exercise 05

Submitted by:
Hitik Panchal (50230156, s17hpanc)
Rouy Alfahel (3389227, s6roalfa)
Orhan Ugur Aydin (3209243, s6oraydi)
Sebastian Sauerbier (2973879, s6sesaue)
Chenfang Wang, (50292921, s59cwang)
Anand Karna (50393435, s03akarn)
Khashayar Alavi (50071735, S76kalav)
Feiyue Cheng (3305865, s6fechen)

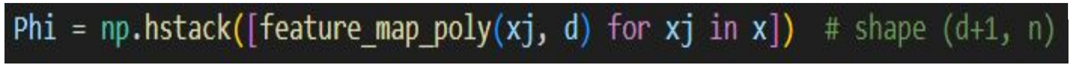
Task 5.1: Regularized Least Squares (Ordinary LS Regression)

Goal: Fit a high degree polynomial (d=9) to noisy training data

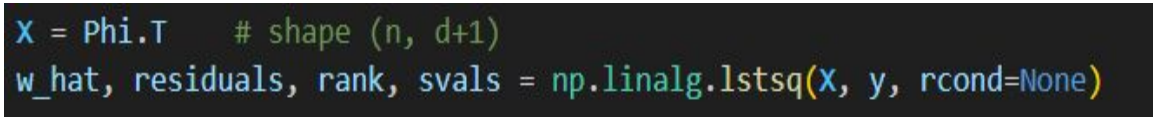
- Model: $f(x) = w^T \varphi(x)$, $\varphi(x) = [1, x, x^2, \ldots, x^d]^T$
- OLS solution: $\hat{w} = (\Phi\Phi^T)^{-1} \Phi y$ (computed via numerically stable least squares)
- Observation: Highly flexible model, strong oscillations → overfitting



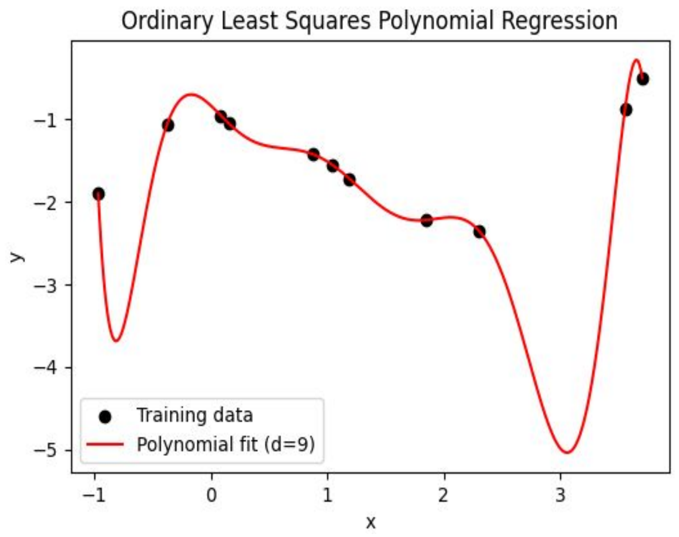
```python
def feature_map_poly(x, d):
    f_map = np.zeros((d + 1, 1), dtype=float)
    for i in range(d + 1):
        f_map[i, 0] = x ** i
    return f_map
```



```python
Phi = np.hstack([feature_map_poly(xj, d) for xj in x])  # shape (d+1, n)
```



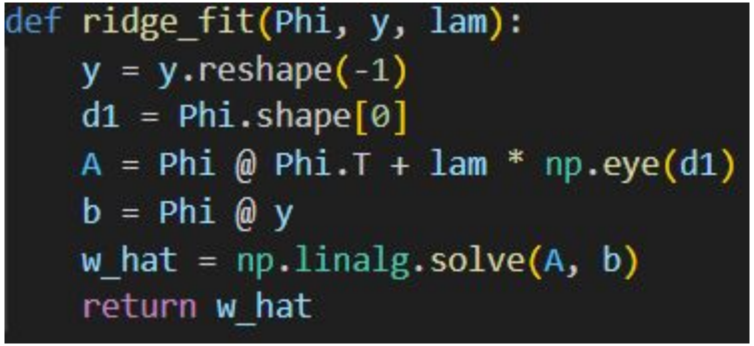
```python
X = Phi.T    # shape (n, d+1)
w_hat, residuals, rank, svals = np.linalg.lstsq(X, y, rcond=None)
```



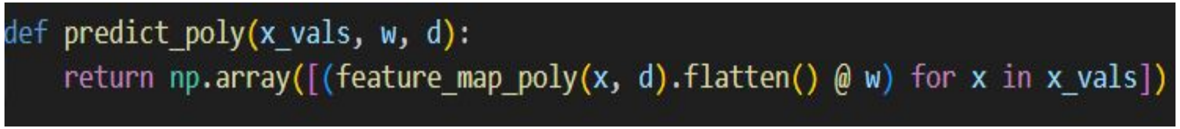
Task 5.1: Regularized Least Squares (Ridge Regression)

Problem: High-degree polynomial is over-parameterized

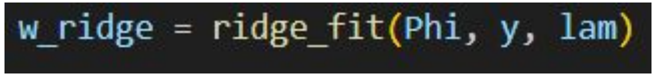
- Regularization can penalize large weights and improves stability
- Ridge solution: $\hat{w} = (\Phi\Phi^T + \lambda I)^{-1} \Phi y$, $\lambda = 0.5$ (L2-Regularization)
- Result: Smoother curve, reduced overfitting, more reasonable model



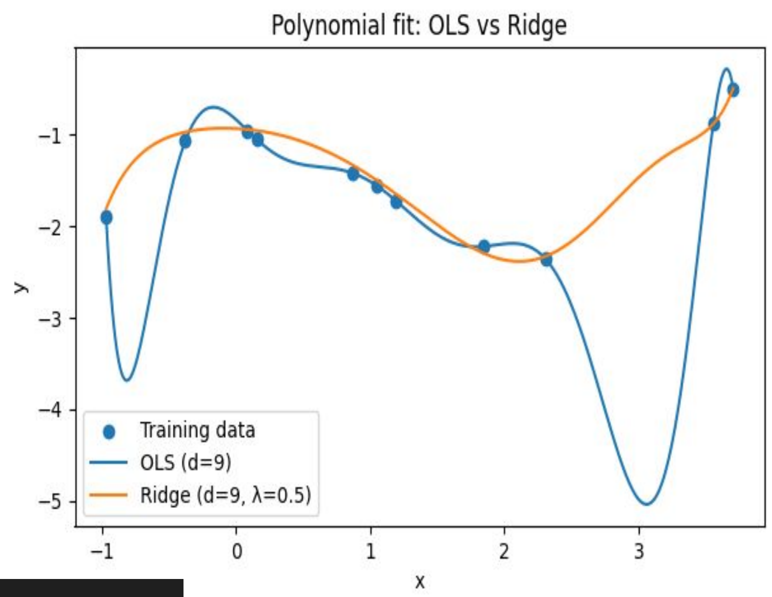
```python
def ridge_fit(Phi, y, lam):
    y = y.reshape(-1)
    d1 = Phi.shape[0]
    A = Phi @ Phi.T + lam * np.eye(d1)
    b = Phi @ y
    w_hat = np.linalg.solve(A, b)
    return w_hat
```



```python
def predict_poly(x_vals, w, d):
    return np.array([(feature_map_poly(x, d).flatten() @ w) for x in x_vals])
```



```python
w_ridge = ridge_fit(Phi, y, lam)
```



Task 5.2: Kernel least squares

Goal: • Fit a smooth function f(x) to noisy training data (x_i, y_i)

• Use a kernel method

• Compare results for degree d = 3 and a higher degree (e.g., d = 9)

• Understand how regularization λ controls smoothness vs. overfitting

What is Kernel Least Squares (Kernel Ridge Regression)?

• Least squares regression with L2 regularization, done in kernel space

• We do not learn feature weights directly; we solve using the kernel matrix

• Output is a smooth function that balances data fit and simplicity (via λ)

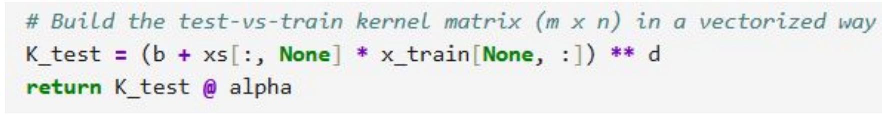
Prediction formula:

• K_ij = k(x_i, x_j)

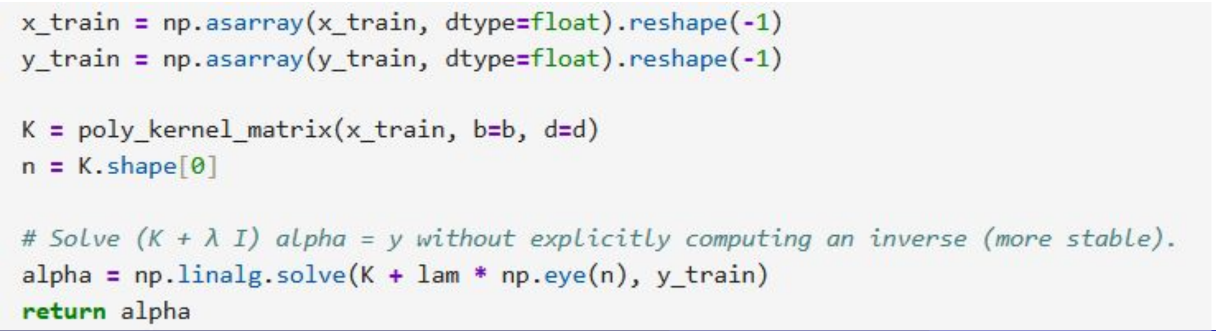
• k(x) = [k(x, x_1), …, k(x, x_n)]^T

• f_hat(x) = k(x)^T (K + λ I)^(-1) y

• λ > 0 reduces overfitting and makes the system stable



```
# Build the test-vs-train kernel matrix (m x n) in a vectorized way
K_test = (b + xs[:, None] * x_train[None, :]) ** d
return K_test @ alpha
```

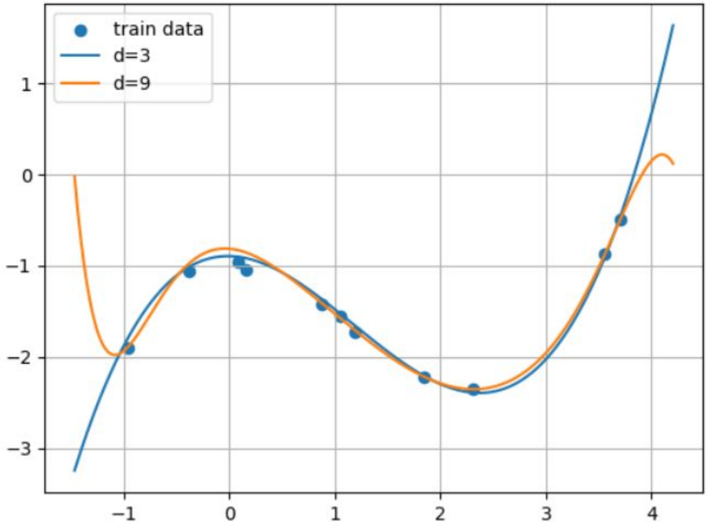


```
x_train = np.asarray(x_train, dtype=float).reshape(-1)
y_train = np.asarray(y_train, dtype=float).reshape(-1)

K = poly_kernel_matrix(x_train, b=b, d=d)
n = K.shape[0]

# Solve (K + λ I) alpha = y without explicitly computing an inverse (more stable).
alpha = np.linalg.solve(K + lam * np.eye(n), y_train)
return alpha
```

Task 5.2: Polynomial kernel + Observations + GP Connection

What we observe (from the plots):

- $d = 3$: smooth cubic-like curve (stable fit)
- $d = 9$: more flexible curve, can behave unstable near boundaries
- Smaller $\lambda \to$ more wiggles (more overfitting)
- Larger $\lambda \to$ smoother curve (less overfitting, more bias)



Polynomial kernel:

- K_ij = (b + x_i * x_j)^d
- k(x)_j = (b + x * x_j)^d
- Parameters: $\lambda = 0.5$, $b = 1$, $d = 3$

Connection to Gaussian Processes (GP):

- Kernel LSQ: f_hat(x) = k(x)^T (K + λ I)^(-1) y
- GP regression posterior mean (noise variance σ_n^2):

E[f(x) | y] = k(x)^T (K + σ_n^2 I)^(-1) y

- Same formula if: λ = σ_n^2 (regularization equals noise variance)
- GP additionally provides uncertainty:

Var(f(x) | y) = k(x,x) - k(x)^T (K + σ_n^2 I)^(-1) k(x)

Task 5.3: Least squares SVMs for regression

Goal: Fit a regression model to the noisy dataset `noisyCubicPoly.csv` to predict $y \in \mathbb{R}$.

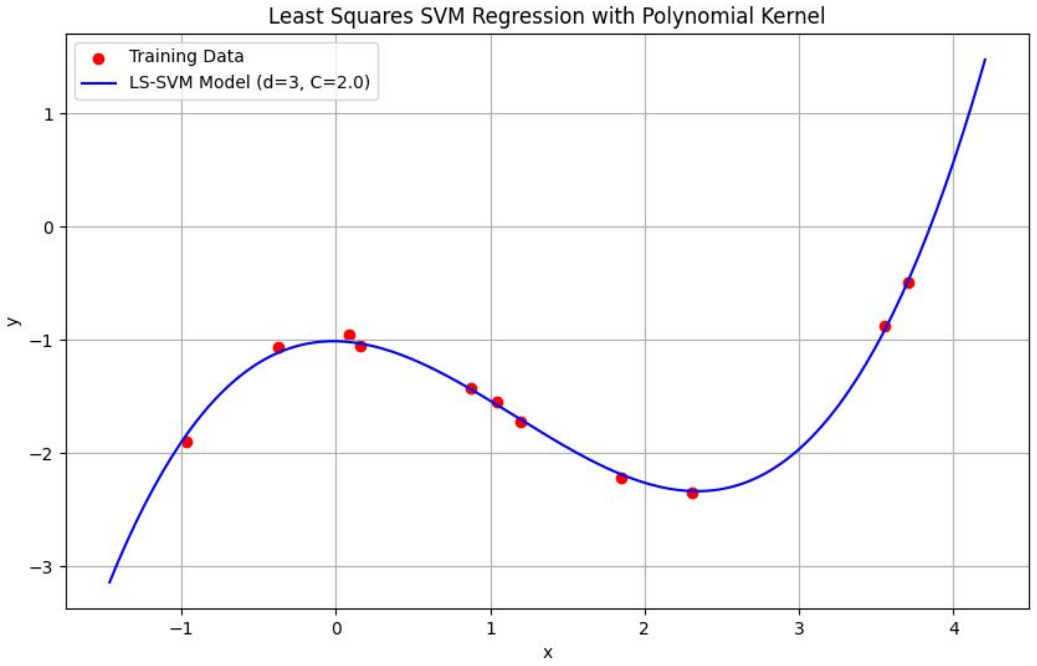
Approach

We employed Least Squares Support Vector Machines (LS-SVM) for regression.

- Formulation: Minimizes the squared error while incorporating regularization.
- Solution: Achieved through a closed-form solution via a system of linear equations, eliminating the need for iterative optimization.
- Kernel: A Polynomial Kernel k(x,z)=(1+xz)^3 was utilized to effectively model non-linear cubic relationships.
- Parameters: The specific parameters used were C=2 (Regularization parameter) and b=1, d=3 (Kernel parameters).

```
# Polynomial Kernel
def poly_kernel(x1, x2):
    return (1.0 + x1 * x2) ** 3.0

# Construct System Matrix (K + 1/C * I)
upper_left = K + (1.0 / 2.0) * np.eye(N)
# Solve Linear System
solution = np.linalg.solve(A, rhs)
lambdas, b_hat = solution[:N], solution[N]
```

Task 5.3: Least squares SVMs for regression



Results

The blue curve represents the fitted model. It successfully captures the underlying cubic trend of the red data points. The model is robust to the noise present in the dataset, providing a smooth approximation.

Conclusion

- The LS-SVM with a polynomial kernel is highly effective for this regression task.
- The choice of d = 3 was appropriate for the cubic data structure.

Learning

- **Kernel Trick:** Allows linear algorithms to solve non-linear problems efficiently.
- **LS-SVM Advantage:** Provides an analytical solution (linear system) which is computationally stable and easy to implement compared to standard SVMs (QP problems).

5.4 Kernel SVMs for Binary Classification

The kernelized L_2 SVM dual problem is

$$\begin{aligned} \min_{\mu \in \mathbb{R}^n} \quad & \mu^\top \Big(K \odot (yy^\top) \;+\; yy^\top \;+\; \frac{1}{C} I \Big) \mu \\ \text{s.t.} \quad & \mathbf{1}^\top \mu = 1, \quad \mu \geq 0 \end{aligned}$$

- Kernel K is polynomial

$$K \in \mathbb{R}^{n \times n}, \quad K_{ij} = k(x_i, x_j), \quad k(x_i, x_j) = (b + x_i^\top x_j)^d$$

Training data

$$\{(x_i, y_i)\}_{i=1}^n, \quad x_i \in \mathbb{R}^2, \quad y_i \in \{-1, +1\},$$

After solving for μ, the decision function is

$$f(x) = \sum_{i \in S} \mu_i y_i k(x_i, x) \;+\; \sum_{i \in S} \mu_i y_i$$

where

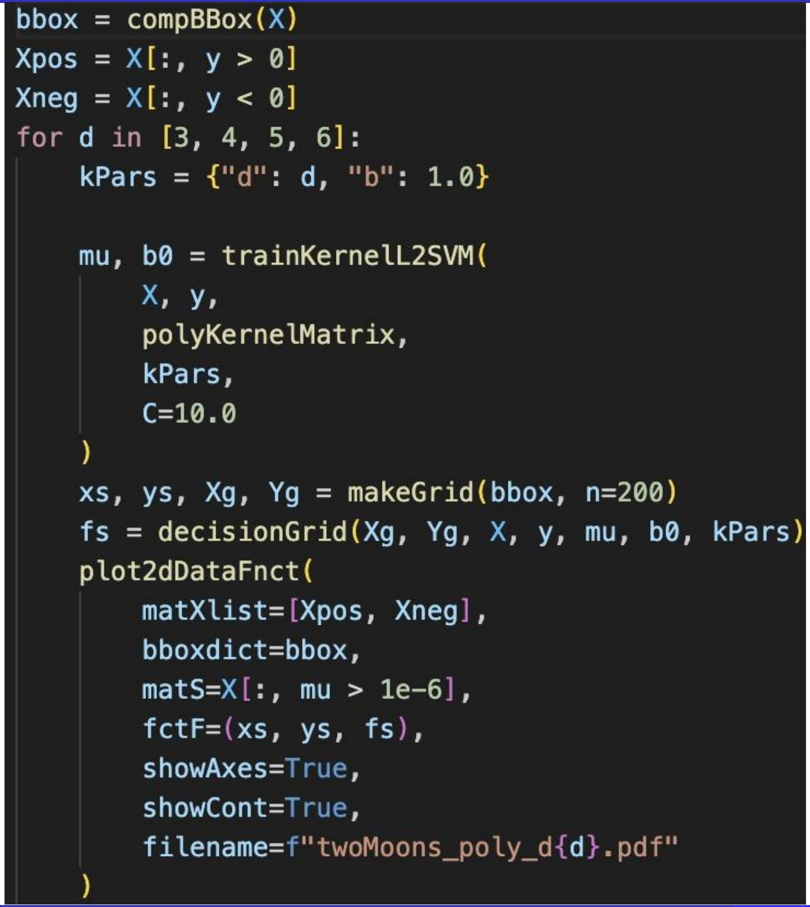
$$S = \{ i \mid \mu_i > 0 \}$$

Prediction rule:

$$\hat{y} = \text{sign}(f(x))$$

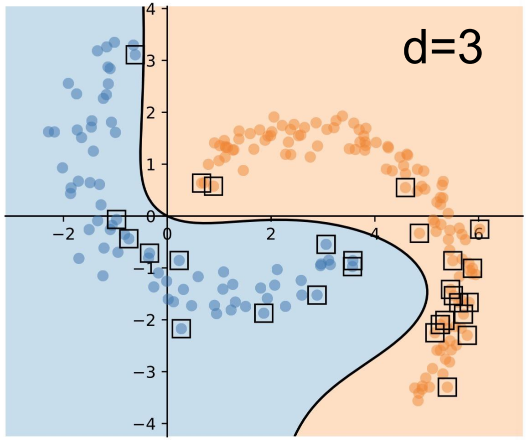
The decision boundary is defined by

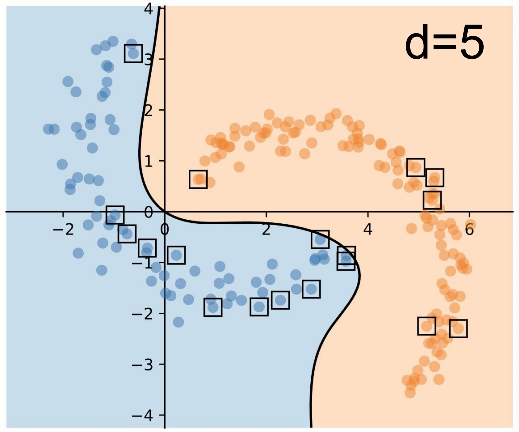
$$f(x) = 0$$

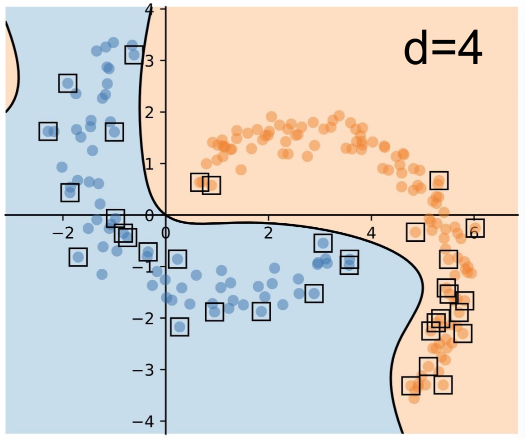


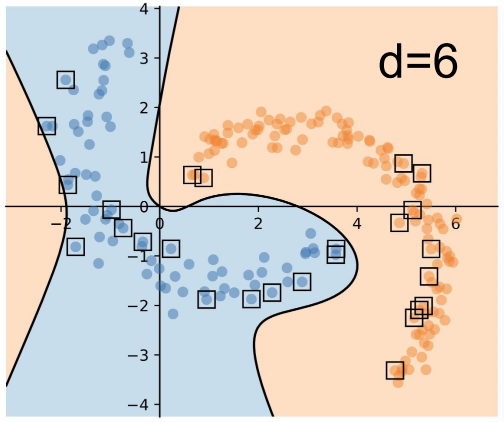
```python
bbox = compBBox(X)
Xpos = X[:, y > 0]
Xneg = X[:, y < 0]
for d in [3, 4, 5, 6]:
    kPars = {"d": d, "b": 1.0}

    mu, b0 = trainKernelL2SVM(
        X, y,
        polyKernelMatrix,
        kPars,
        C=10.0
    )
    xs, ys, Xg, Yg = makeGrid(bbox, n=200)
    fs = decisionGrid(Xg, Yg, X, y, mu, b0, kPars)
    plot2dDataFnct(
        matXlist=[Xpos, Xneg],
        bboxdict=bbox,
        matS=X[:, mu > 1e-6],
        fctF=(xs, ys, fs),
        showAxes=True,
        showCont=True,
        filename=f"twoMoons_poly_d{d}.pdf"
    )
```

5.4 Kernel SVMs for Binary Classification









- Polynomial kernels are globally sensitive: a data point influences the decision function everywhere, whereas Gaussian kernels are locally sensitive because their influence decays rapidly with distance.Therefore, increasing the polynomial degree d leads to more curved boundary.

- When the degree d is even, the polynomial kernel is less sensitive to the sign of inner products, i.e. to the relative direction of data points. This can induce symmetric decision patterns. For data such as the two-moons, where directional asymmetry matters, odd degrees may be more suitable.

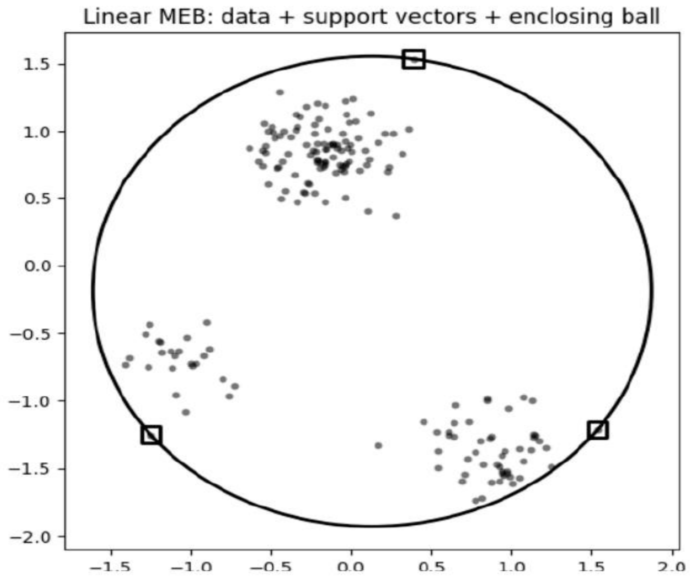
Task 5.5.1: Dual MEB - Algorithm and Result

Which algorithm "screams" to be used?

- The dual is a convex quadratic program on the simplex ($\mu \geq 0$, $\sum_i \mu_i = 1$) $\Rightarrow$ use a **QP solver**.
- **Concrete algorithm used: SLSQP** (Sequential Least Squares Programming) - solves the quadratic objective with equality + inequality constraints directly.

What we see on threeBlobs

- The enclosing ball is determined by only a **few boundary points**.
- These show up as **support vectors** (squares); most points are inside and do not affect the radius.



Linear MEB with highlighted support vectors

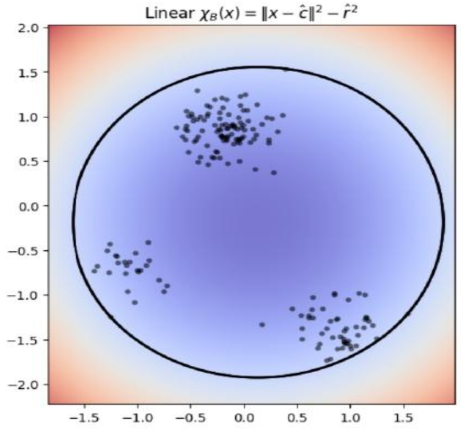
Task 5.5.2 & 5.5.3: χ_B figure + Kernel MEB Observations

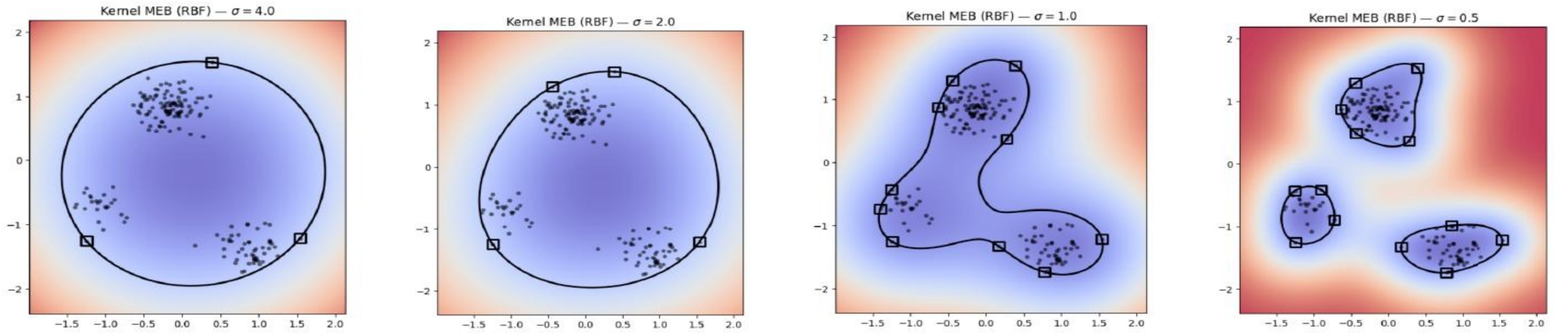
5.5.2: $\chi_B(x)$ on a grid

- We plotted $\chi_B(x)$ and observed the expected picture: **blue inside**, **red outside**, and the **black contour at $\chi_B(x) = 0$** matches the MEB boundary.

5.5.3: Kernel MEB (RBF) - what changes with σ

- **Large σ (4, 2):** smooth/global influence $\Rightarrow$ boundary looks close to the linear ball.
- **Smaller σ (1, 0.5):** local influence $\Rightarrow$ boundary bends toward clusters and forms separate bulges.
- Support vectors still mark where the boundary is "pulled" outward.


Linear $\chi_B(x) = \|x - \hat{c}\|^2 - \hat{r}^2$
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Kernel MEB (RBF) — $\sigma = 4.0$
Kernel MEB (RBF) — $\sigma = 2.0$
Kernel MEB (RBF) — $\sigma = 1.0$
Kernel MEB (RBF) — $\sigma = 0.5$