7. Implementing SVMs

In [18]:
```
# !pip install ucimlrepo
```

In [2]:
```
from ucimlrepo import fetch_ucirepo
import pandas as pd
import numpy as np
import cvxpy as cp
from matplotlib import pyplot as plt
```

In [3]:
```
# !! DO NOT MODIFY THIS CELL !!

# Download and preprocess the dataset.
# fetch dataset
heart_disease = fetch_ucirepo(id=45)
X = heart_disease.data.features
# Convert categorical features into one-hot encode
categorical_features = ['cp','thal','slope','restecg']
X = pd.get_dummies(X, columns=categorical_features)

y = heart_disease.data.targets
print(f"Number of samples in all full dataset is: {len(X)}.")

# Check if our train set has missing value
na_in_features = X.isna().any(axis=1).sum()
na_in_trainY = y.isna().sum()
print(f"Number of rows with missing values in features: {na_in_features}")

# Drop the rows with missing values.
indices_with_nan = X.index[X.isna().any(axis=1)]
X = X.drop(indices_with_nan)
y = y.drop(indices_with_nan)

# Divide train/test
np.random.seed(6464)
msk = np.random.rand(len(X)) < 0.75
X_train = X[msk]
X_test = X[~msk]
y_train = y[msk]
y_test = y[~msk]

# Convert problem to binary problem
X_train = np.array(X_train,dtype='float')
X_test = np.array(X_test,dtype='float')
y_train = np.array([-1 if i==0 else 1 for i in y_train.values],dtype='float')
y_test = np.array([-1 if i==0 else 1 for i in y_test.values],dtype='float')

print(f"Shapes: X_train: {X_train.shape}, y_train: {y_train.shape}, X_test: {X_t
```

```
Number of samples in all full dataset is: 303.
Number of rows with missing values in features: 4
Shapes: X_train: (216, 22), y_train: (216,), X_test: (83, 22), y_test: (83,)
```

In [4]:
```
# Normalize X_train and X_test using the statistics of X_train.
# 1. Compute the mean and standard deviation for each feature in X_train
# 2. Subtract the mean from each feature and divide by the standard deviation
#    for both X_train and X_test.
```

```
mean_X_train = np.mean(X_train,axis=0)
std_X_train = np.std(X_train,axis=0)

X_train_normalized = (X_train - mean_X_train)/std_X_train
X_test_normalized = (X_test - mean_X_train)/std_X_train
```

Why use train mean and std

- **Prevent Data Leakage:** Using test data for normalization would leak information, leading to overly optimistic results.
- **Real-World Consistency:** In deployment, only training data statistics are available, so the model must generalize based on them.
- **Fair Evaluation:** Ensures test data follows the same distribution as training data for an unbiased assessment.

In [5]:
```
# Print the mean and standard deviation of the first and last feature.
print("X_train First Feature")
print("----------------------------")
print(f"Mean: {mean_X_train[0]:.2f}")
print(f"Standard deviation: {std_X_train[0]:.2f} \n")


print("X_train Last Feature")
print("----------------------------")
print(f"Mean: {mean_X_train[-1]:.2f}")
print(f"Standard deviation: {std_X_train[-1]:.2f}")
```

```
X_train First Feature
----------------------------
Mean: 54.99
Standard deviation: 9.08

X_train Last Feature
----------------------------
Mean: 0.50
Standard deviation: 0.50
```

In [6]:
```
# Train SVM

# Complete the `trainSVM` function to find the optimal w and b that minimize
# the primal SVM objective given in the write-up.
# The function takes three inputs:
# - trainX: the normalized train features with shape (#train_samples, #features)
# - trainY: train labels with shape (#train_samples,)
# - C: C parameter of the minimization problem
# The function should return a three-tuple with:
# - w: the weight vector with shape (#features,)
# - b: the bias. A scalar with shape (1,)
# - xi: the slack variables with shape (#train_samples,)

# You can use cvxpy that we imported as cp
# You may find cp.Variable, cp.Minimize, cp.Problem useful
# For the problem solver, prefer the default, cp.CLARABEL

def trainSVM(X_train_normalized, y_train, C):
    n_features = X_train_normalized.shape[1]

    # Define variables
```

```
    w = cp.Variable(n_features)
    b = cp.Variable()
    xi = cp.Variable(X_train_normalized.shape[0])

    constraints = [
    # SVM constraint
    cp.multiply(y_train, (X_train_normalized @ w + b)) >= 1 - xi,
    # Slack variables must be non-negative
    xi >= 0
    ]

    # Objective function
    objective = cp.Minimize(0.5 * cp.norm(w, 2)**2 + C * cp.sum(xi))

    # Solve the problem
    problem = cp.Problem(objective, constraints)
    problem.solve()

    return w.value, b.value, xi.value
```

In [7]:
```
# Solve SVM with C = 1 and print the first three weights, b and the first
# three slack variables as instructed in the write-up
w_C1, b_C1, xi_C1 = trainSVM(X_train_normalized, y_train, C=1)

print(f"First 3 weights:\n{w_C1[:3]}")
print(f"Bias: {b_C1:.4f}")
print(f"First 3 slack variables:\n{xi_C1[:3]}")
```

```
First 3 weights:
[-0.01280084  0.51706872  0.27813637]
Bias: 0.0811
First 3 slack variables:
[-1.70119328e-10 -1.64395885e-10 -1.69587409e-10]
```

In [8]:
```
# Solve SVM with C = 0 and print the first three weights, b and the first
# three slack variables as instructed in the write-up
w_C0, b_C0, xi_C0 = trainSVM(X_train_normalized, y_train, C=0)

print(f"First 3 weights:\n{w_C0[:3]}")
print(f"Bias:{b_C0:.4f}")
print(f"First 3 slack variables:\n{xi_C0[:3]}")
```

```
First 3 weights:
[ 3.09523259e-06 -8.18802636e-06 -9.46615246e-06]
Bias:-10.4476
First 3 slack variables:
[429.58840105 434.02071004 414.34670026]
```

Explanation of the different C outputs

This difference arises because **C controls the trade-off between maximizing the margin and minimizing classification errors**. With **C = 0**, the optimization focuses only on maximizing the margin without penalizing misclassified points, allowing larger slack variables (ξ). However, when **C > 0**, the term (C \sum \xi_i) is introduced in the objective function, enforcing a stricter penalty for misclassification, thus reducing slack values.

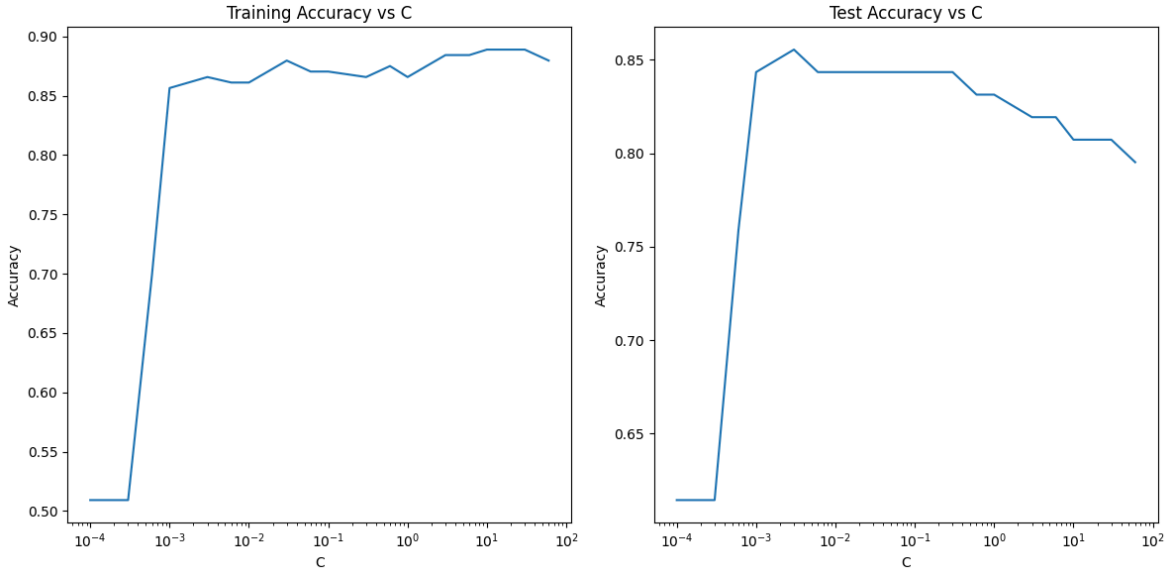
This explains why we introduce the **(C \sum \xi_i)** term in Soft-SVM—it ensures that we balance margin maximization with classification accuracy. Without it, the model would

ignore classification errors entirely.

In [9]:
```python
# Eval SVM

# Write a function to evaluate the SVM model given its `w` and `b` parameters
# on evaluation data `X_eval` and true labels `y_eval`.
#  1. Estimate the labels of `X_eval`.
#  2. Return the ratio of accurately estimated labels by comparing with `y_eval`
def evalSVM(X_eval, y_eval, w, b):
    y_pred = np.sign(np.dot(X_eval, w) + b)
    accuracy = np.mean(y_pred == y_eval)
    return accuracy
```

In [14]:
```python
train_accuracies = []
test_accuracies = []
C_values = []

# Given C values based on homework specifications
a_values = [1, 3, 6]
q_values = [-4, -3, -2, -1, 0, 1]
C_possibilities = [a * (10 ** q) for a in a_values for q in q_values]

for C in C_possibilities:
    w, b, _ = trainSVM(X_train_normalized, y_train, C)
    train_accuracy = evalSVM(X_train_normalized, y_train, w, b)
    test_accuracy = evalSVM(X_test_normalized, y_test, w, b)

    C_values.append(C)
    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)

# Sort lists based on C_values while keeping corresponding values aligned
sorted_data = sorted(zip(C_values, train_accuracies, test_accuracies), key=lambd
C_values, train_accuracies, test_accuracies = zip(*sorted_data)

# Convert tuples back to lists
C_values = list(C_values)
train_accuracies = list(train_accuracies)
test_accuracies = list(test_accuracies)
```

In [15]:
```python
# Plotting and reporting the desired values
plt.figure(figsize=(12,6))
plt.subplot(1, 2, 1)
plt.plot(C_values, train_accuracies)
plt.xscale('log')
plt.title('Training Accuracy vs C')
plt.xlabel('C')
plt.ylabel('Accuracy')

plt.subplot(1, 2, 2)
plt.plot(C_values, test_accuracies)
plt.xscale('log')
plt.title('Test Accuracy vs C')
plt.xlabel('C')
plt.ylabel('Accuracy')

plt.tight_layout()
plt.show()
```



In [16]:
```
# Desired values
best_train_C = C_values[np.argmax(train_accuracies)]
best_train_accuracy = max(train_accuracies)

print(f"Best Training C: {best_train_C}\nAccuracy: {best_train_accuracy:.2f}")
```

```
Best Training C: 10
Accuracy: 0.89
```

In [17]:
```
best_test_C = C_values[np.argmax(test_accuracies)]
best_test_accuracy = max(test_accuracies)


print(f"Best Test C: {best_test_C}\nAccuracy: {best_test_accuracy:.2f}")
```

```
Best Test C: 0.003
Accuracy: 0.86
```

In [ ]:
```

```

mbaraka

1) Naive Bayes

a) Number of free parameters Θ and Π that is to be estimate use N-B

→ Estimating Π

For each class of $Y = i$:

$$\Pi_i = P(Y = i)$$

$$\therefore \Pi_1 + \Pi_2 + \Pi_3 + \Pi_4 = 1$$

$$\therefore 4 - i = 3$$

∴ 3 parameters of Π

→ Estimating for Θ $[X_n - 1] \times$ no. of class

For features X_1 :→ $(2-1) \times 4 = 4$

X_2 :→ $(3-1) \times 4 = 8$

X_3 :→ $(4-1) \times 4 = 12$

X_4 :→ $((5-1) \times 4 = 16$

$$\sum X_j = 4 + 8 + 12 + 16 = 40$$

∴ For $\Theta + \Pi$ $= 40 + 3$

$= 43$ parameters

b) Number of free parameters to be estimated if the features are independent conditioned on the label.

Estimat entire distribution $P(X_1, X_2, X_3, X_4 | Y)$

→ Each feature has different number of possible values

$X_1 \in [1, 2]$, $X_2 \in [1, 2, 3]$, $X_3 \in [1, 2, 3, 4]$, $X_4 \in [1, 2, 3, 4, 5]$

∴ Total numb. per class

$\Rightarrow 2 \times 3 \times 4 \times 5 = 120$

For 4 class $\Rightarrow 120 \times 4 = 480$

But free parameters per class

$120 - 1 = 119 \times 4$ class $= 476$

$= 476$

Total $= 476 + 3 = 479$

C) Advantage of ~~using~~ assuming conditional independence

→ In part (b) where we do not assume conditional independe, we need to estimate 479 parameter compared to 43 in part (a), ~~the~~ ~~gener~~ this indicates an advantage of reduction of the computational cost and improved efficiency in learning.

mbaruta

② Naive Bayes in Practice

a) Estimate $Pr(y=1)$ | positive reviews, $Pr(y=2)$ | neutral reviews, $Pr(y=3)$ | negative reviews,

$$Pr(y=c) = \frac{\text{Number of reviews in } C}{\text{Total number of reviews}}$$

$Pr(y=1) = 4/8 = 0.5$

$Pr(y=2) = 2/8 = 0.25$

$Pr(y=3) = 2/8 = 0.25$

b) Feature vector X for each positive review in the training set.

V = {1: "incredible", 2: "plot", 3: "great", 4: "amazing", 5: 'okay', 6: "decent" 7: "movie", 8: "no", 9: 'acting', 10: "waste"}

→ "great movie amazing"
[0 0 1 1 0 0 1 0 0 0]

→ "incredible movie"
[1 0 0 0 0 0 1 0 0 0]

→ "great acting amazing plot"
[0 1 1 1 0 0 0 0 1 0]

c) N.B likelihood of a sentence feature x given class C is:

$$Pr(X|y=c) = \prod_{k=1}^{n} (\theta_{c,k})^{x,k}$$ | $\theta_{c,k}$ ⇒ weight of word k in class C

Calculate the maximum likelihood of:

$$\theta_{c,k} = \frac{\text{word } k \text{ in Class } C}{\text{Total count of words in class } c}$$

→ $\theta_{1,4}$ = Positive review | amazing = 4/13

→ $\theta_{1,7}$ = +ve review | movie = 2/13

→ $\theta_{2,4}$ = neutral | amazing = 1/6

→ $\theta_{2,7}$ = Neutral | movie = 1/6

$\theta_{3,4}$ = -ve review | amazing = 1/5

$\theta_{3,7}$ = -ve review | movie = 1/5

d) New review "amazing movie" decide whether +ve, neutral or -ve. based on Naive Bayes Classifier, learned from above data.

Class 1: positive

Prior prob $= \frac{4}{8} = 0.5$

$\theta_{1,4} = \frac{4}{13}$

$\theta_{1,7} = \frac{2}{13}$

$\therefore = \frac{4}{13} \times \frac{2}{13} \times 0.5$

$= 0.00237$

$= 2.37\%$

Class 2: neutral

Prior $= \frac{2}{8} = 0.25$

$\theta_{2,4} = \frac{1}{6}$

$\theta_{2,7} = \frac{1}{6}$

$= \frac{1}{6} \times \frac{1}{6} \times 0.25$

$= 0.00694$

$= 0.694\%$

Class 3:

Prior $= \frac{2}{8} = 0.25$

$\theta_{3,4} = \frac{1}{5}$

$\theta_{3,7} = \frac{1}{5}$

$= \frac{1}{5} \times \frac{1}{5} \times 0.25$

$= 0.01$

$= 1\%$

Since +ve review has the highest probability

$\therefore$ positive review

e) Use laplacian smoothing with $\alpha = 1$ to decide whether the review "desert movie" is +ve, neutral or -ve. Describe the problem we will encounter if we had not used laplacian smoothing.

Using Laplacian

$$\theta_{c,k} = \frac{\#\text{ of word}_k \text{ in } C + \alpha}{\#\text{ of words in } C + \alpha \times \text{unique words}}$$

Class 1: Positive

Prior $= 0.5$

$\theta_{4,de} = \frac{0+1}{13+1(10)} = \frac{1}{23}$

$\theta_{1,7} = \frac{2+1}{13+1(10)} = \frac{3}{23}$

$\Rightarrow \frac{1}{23} * \frac{3}{23} \times 0.5 =$

$= 0.00222$

$= 0.222\%$

$\therefore$ Neutral review

Class 2: Neutral

prior $= 0.25$

$\theta_{4,des} = \frac{0+1}{6+1(10)} = \frac{1}{16}$

$\theta_{2,7} = \frac{1+1}{6+1(10)} = \frac{2}{16}$

$\Rightarrow \frac{1}{16} * \frac{1}{8} \times 0.25$

$= 0.00097$

$= 0.097\%$

$= 0.78$

Class 3: negative

Prior $= 0.25$

$\theta_{3,des} = \frac{0+1}{5+(10)} = \frac{1}{15}$

$\theta_{3,7} = \frac{1+1}{5+(10)} = \frac{2}{15}$

$\Rightarrow \frac{1}{15} \times \frac{2}{15} \times 0.25 = 0.0044$

$= 0.44\%$

$\Rightarrow$ Neutral review

Connect

→ Without laplacian smoothing, words that do not appear in a given class will have a zero probability hence it will cause the entire likelihood of being in positive class and negative class to be zero and this will prevent classification.

3> Logistic Regression

a) Is it possible to get closed form for the parameter $\hat{w}$ that Maximizes the log likelihood?

=> No;

If not, how would you compute $\hat{w}$ in practice?

=> Will use gradient descent to iteratively update w.

Explain method to find $\hat{w}$ in few sentences to give a short Pseudo code

General Approach is:

1) Initialize the weights $\hat{w}$

2) Compute the gradient of the log likelihood function by;

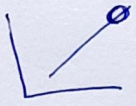
$$\nabla L(w) = \sum_{i=1} y_i - P(y_i|x_i, w))x_i$$

3) Update the weights using the gradient [$w \leftarrow w + \eta \nabla L(w)$]

4) The process is repeated until it convergence.

b) Find decision bundary, which is the set of x satisfying

$P(y=1|x,w) = P(y=0|x,w)$



$$P(y=1|x,w) = \frac{1}{1+\exp(-w^T x)}$$

$$P(y=0|x,w) = 1 - P(y=1|x,w)$$

$$\therefore \frac{1}{1+\exp(-w^T x)} = 1 - \frac{1}{1+\exp(-w^T x)} \quad \times 1+\exp(-w^T x)$$

$$1 = 1 + \exp(-w^T x) - 1$$

$$1 = \exp(-w^T x)$$

$$\log(1) = \log(\exp(-w^T x))$$

$$0 = -w^T x$$

$$\Rightarrow w^T x = 0$$

2

c) Is this model a linear classifier?

→ Yes, since the decision boundary is given by $w^T x = 0$ which is a linear equation, therefore it is a linear classifier.

d) Model has high tolerance on FN which is costly, how do we adjust the model to reduce False Negatives

→ False Negatives can be by adjusting the decision boundary threshold: $P(y=1 \mid x, w) \geq t$, reducing it will increase its sensitivity to detect positives.

∴ I could adjust the threshold if $y=1$;

$$P(y=1 \mid x, w) \geq 0.25$$

4) Solving Logistic Regression

Eq (3) $\ell(w) = -\sum_{i=1}^{n} \left(y_i w^T x_i - \log\left(1+\exp(w^T x_i)\right)\right)$ .... (Eq 3)

a) Starting from definition of negative log-likelihood [eq (4)], show that it is equal to the cross-entropy log (Eq. 3)

$$-\ell(w) = -\log\left(\prod_{i=1}^{n}\left(\frac{1}{1+\exp(-w^T x)}\right)^{y_i}\left(1-\frac{1}{1+\exp(-w^T x_i)}\right)^{1-y_i}\right)$$

$$= -\sum_{i=1}^{n}\left[y_{(i)} \log\left(\frac{1}{1+\exp(-w^T x_i)}\right) + (1-y_i)\log\left(1-\frac{1}{1+\exp(-w^T x_i)}\right)\right]$$

$$= \overset{\text{Part 1}}{=} y_i \log\left(\left(1+\exp(-w^T x_i)\right)^{-1}\right. \qquad = -y_{(i)} \log\left(1+\exp(-w^T x_i)\right)$$

Part 2

$$1-y_i \quad 1-\frac{1}{1+\exp(-w^T x_i)} = \frac{1+\exp(-w^T x_i)-1}{1+\exp(-w^T x_i)}$$

$$= \frac{\exp(-w^T x_i)}{1+\exp(-w^T x_i)}$$

$$\log\left[\frac{\exp(-w^T x_i)}{1+\exp(-w^T x_i)}\right] = -w^T x_i - \log\left(1+\exp(-w^T x_i)\right)$$

Multiply by $(1-y_i)$

$$= (1-y_i)\left(-w^T x_i - \log\left(1+\exp(-w^T x_i)\right)\right)$$

$$= -(1-y_i)(w^T x_i) - (1-y_i)\log\left(1+\exp(-w^T x_i)\right)$$

Combine

$$= -y_i$$

$$= +\sum_{i=1}^{n}\left[+y_i \log\left(1+\exp\left(-w^T x_i\right)\right) - (1-y_i) w^T x_i - (1-y_i)\log\left(1+\exp\left(-w^T x_i\right)\right)\right]$$

Rewrite

$$= \sum_{i=1}^{n}\left[y_i \log\left(1+\exp\left(-w^T x_i\right)\right) + (1-y_i)\left(-w^T x_i\right) + (1-y_i)\log\left(1+\exp\left(-w^T x_i\right)\right)\right]$$

log factor out $\log\left(1+\exp\left(-w^T x_i\right)\right)$

$$L(w) = \sum_{i=1}^{n}\left[\log\left(1+\exp\left(-w^T x_i\right)\right) - y_i w^T x_i\right]$$

rewrite $-\exp\left(-w^T x_i\right)$ as $\exp\left(w^T x\right)$

$$L(w) = -\sum_{i=1}^{n}\left[y_i w^T x_i - \log\left(1+\exp\left(w^T x_i\right)\right)\right]$$

∴ this is equal to eq (3)

$$L(w) = -\sum_{i=1}^{n}\left[y_i w^T x_i - \log\left(1+\exp\left(w^T x_i\right)\right)\right]$$

b) Show that the negative log-likelihood is a convex function

Negative log likelihood $$\ell(w) = -\sum_{i=1}^{n} \left[y_i w^T x_i - \log\left(1 + e^{w^T x_i}\right) \right]$$

1) First derivative

$$\nabla \ell(w) = \sum_{i=1}^{n} \left(-y_i w^T x_i + \frac{e^{w^T x_i}}{1 + e^{w^T x_i}} x_i \right)$$

Since $$P(y_i = 1 | x_i, w) = \frac{e^{w^T x_i}}{1 + e^{w^T x_i}}$$

$$\therefore \nabla \ell(w) = \sum_{i=1}^{n} \left(P(y_i = 1 | x_i, w) - y_i w^T x_i \right)$$

2) Second derivative

$$\nabla^2 \ell(w) = \sum_{i=1}^{n} P(y_i = 1 | x_i, w)\left(1 - P(y_i = 1 | x_i, w)\right) x_i x_i^T$$

$$\therefore p_i = P(y_i = 1 | x_i, w) = \frac{1}{1 + e^{-w^T x_i}}$$

$$W = \text{diag}(p_i(1 - p_i))$$

$$\therefore \nabla^2 \ell(w) = X^T W X$$

⇒ Show that Hessian is Positive semi-definite

Show that $v^T \nabla^2 \ell(w) v \geq 0$

Substituting Hessian:

$$v^T X^T W X v = (Xv)^T W (Xv)$$

This follows: $v^T \nabla^2 \ell(w) v \geq 0$

$\therefore \nabla^2 \ell(w)$ is positive semi-definite hence $\ell(w)$ is convex.

9) Proving the iterative weighted Least Square update Rule

i) Newton - Raphson Update Rule

$$w_{k+1} = w_k - (\nabla^2 l(w_k))^{-1} \nabla l(w_k) \quad \text{--- (1)}$$

$\rightarrow \nabla l(w) = -X^T(y - p_k)$

$\rightarrow \nabla^2 l(w) = X^T W_k X$

Prove that:- $w_{k+1} = (X^T W_k X)^{-1} X^T W_k z_k$

Substitute $\nabla l(w)$ & $\nabla^2 l(w)$ to (1)

$$\Rightarrow w_{k+1} = w_k - (X^T W_k X)^{-1} (-X^T(y - p_k))$$

$$= w_k + (X^T W_k X)^{-1} (X^T(y - p_k))$$

$$z_k = X w_k + W_k^{-1}(y - p_k)$$

Substitute to the ~~iterative Weighted Least Square~~ update equation

$$\Rightarrow w_{k+1} = (X^T W_k X)^{-1} X^T W_k z_k$$

5) SVMs Hinge Loss and Mistake bounds

Eq 9 $\ell((x,y),w) = \max[0, 1 - y w^T x]$,

Eqn (10) $L(w) = \frac{1}{N} \sum_{i=1}^{N} \ell((x_i, y_i) w) + \lambda \|w\|_2^2$

a) W have a correct prediction of y_i with x_i, i.e $y_i = \text{sign}(w^T x_i)$ What range of values can the hinge loss, $\ell((x,y),w)$ take on this correctly classified example?

Correct prediction: $y_i w^T x > 0$

If correct prediction then;

i) if $y_i w^T x \geq 1$

$\therefore \ell((x_i, y_i), w) = \max[0, 1 - y_i w^T x_i] = 0$

ii) if $0 < y_i w^T x < 1$ correct but within a margin

$\therefore \ell((x_i, y_i), w) = 1 - y_i w^T x_i > 0$

$\therefore$ Possible range;

$\ell((x_i, y_i), w) \in [0, 1]$

What is the possible range of hinge loss for an incorrectly classified example?

Incorrect means; $y_i \neq \text{sign}(w^T x_i)$

$y = 1 \quad w^T x < 0$

$y_i = -1 \quad w^T x_i > 0$

$\therefore y_i w^T x_i < 0 \Rightarrow$ negative values in the hinge

$\therefore \ell((x_i, y_i), w) = \max(0, 1 - y_i w^T x_i) = 1 - y_i w^T x_i > 1$

$\therefore$ Possible range;

$\ell((x_i, y_i), w) \geq 1$

b) Upper bound for the mistake made by using Hinge loss.

Show that

$$\frac{1}{N} M(w) \leq \frac{1}{N} \sum_{i=1}^{N} \max[0, 1 - y_i w^T x_i]$$

where $M(w)$ - number of mistakes made when we use the weight vector w to classify dataset. where $y_i \neq \text{sgn}(w^T x_i)$

Generally

$$M(w) = \sum_{i=1}^{n} 1\{y \neq \text{sgn}(w^T x_i)\}$$

; $1(\cdot) \Rightarrow$ is indicator func $= 1$ if predictor is incorrect.

$y = 1$ & $w^T x_i < 0$

$y_i = -1$ $w^T x_i > 0$

For single predictor

$$\ell((x_i, y_i), w) = \max(0, 1 - y_i w^T x_i)$$

for incorrect predict

$$y_i w^T x_i < 0$$

$\therefore$ the means

$$\ell((x_i, y_i), w) = 1 - y_i w^T x_i \geq 1$$

Average mistakes $= \frac{M(w)}{N}$

for all data points $= \sum_{i=1}^{N} \ell((x_i, y_i), w)$

From part (a) when $y_i \neq \text{sgn}(w^T x_i)$ Hinge is atleast 1

$\therefore M(w) \leq \sum_{i=1}^{n} \ell((x_i, y_i), w)$

Averaging by N

$$\therefore \frac{M(w)}{N} \leq \frac{1}{N} \sum_{i=1}^{n} \ell((x_i, y_i), w)$$

Substitute

$$\frac{1}{N} M(w) \leq \frac{1}{N} \sum_{i=1}^{n} \max[0, 1 - y_i w^T x_i]$$

c) When data is separable, the minimize my classifier some training samples. How should we adjust λ for SVM to work properly on data?

Increas the λ

Why?

When λ is increased, the regularizeting weight that Penalizes larger weights in the hinge loss is also increased. This pushes the model to focus on creating a large margin that will push the model to have fewer misclassifications.

6) Support Vector Machine:- Slack variable & Duality Intuition

Eqn (2 ⇒ Hard-SVM)

$$\min_{w,b} \frac{1}{2}\|w\|_2^2 \quad s.t \quad y_i(w^T x_i + b) \geq 1, \quad \forall i \in [n]$$

a) Why is hard SVM formulation may fail in real word.

→ Hard SVM formulation assumes that classes are linearly separable with a clear margin between them which is not the case in real-word datasets; due to noise in data, overlapping classes or outlies it is difficult to separate the dataset linearly. This assumption leads it to choose a poor decision boundary that misclassifies.

b) Soft SVM with slack variable for each data pt

$$\min_{w,b,\xi_i-\xi_n} \left\{ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} \xi_i \right\}$$

$$s.t \; \forall i, \; y_i(w^T x_i + b) \geq 1 - \xi_i, \; \xi_i \geq 0$$

How does introduction of slack ξ_i resolve the limitation of hard-SVM.

The slack variable ξ_i is introduce to give / allow some flexibility ~~to~~ for some points to fall within the margin or misclassified. The model does some mistakes that hard SVM had constrains on this allows some points be on the wrong side or class which is easy to work with real world data.

Why use "$\geq 1-\xi_i$" instead of "≥ 1"

→ This allows points to fall within the margin or misclassified by some amount ~~of~~ captured by the slack ξ_i

Why include "$C\sum_i \xi_i$" :→ The function maximizes the margin minimising $\|w\|^2$ and penalize the model for misclassification.

c) Dual formulation of the soft-SVM:

Primal

$$\min_{w,b,\xi_1 \dots \xi_n} \left\{ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} \xi_i \right\} \quad \text{s.t } \forall i, y_i(w^T x_i + b) \geq 1 - \xi_i, \xi_i \geq 0$$

Dual

$$\max_{\alpha} \left\{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j k(x_i, x_j) \right\}$$

$$\text{s.t } \forall i,\ 0 \leq \alpha_i \leq C,\ \sum_{i=1}^{n} \alpha_i y_i = 0$$

dual variable are represented by α_i and $k(x_i, x_j) = x_i^T x_j$.

⇒ Find the number of variables to be optimized in primal & dual forms of SVM.

Primal

weight w = dimension of features d in training data.

bias b = 1 scaler variable

slack variable ξ_i = n slack variable; n is number of training samples

∴ Total = $d + 1 + n$

Dual

α_i : n dual variable for each training data.

Total = n

⇒ If we change kernel, does this change the number of dual variables we need?

→ No, the number of variables will still remain n.

d) Why might you prefer SVM in dual than primal?

→ Efficiency: since dual allows us to work on higher dimension

→ Flexibility with kernel: dual accomodate various kernel functions.

8. Non-linear Basis Function

① What is the basis function $\phi(x)$ for this problem

$y_i = \omega_0 + \omega_1 \sin(x_i) + \omega_2 \cos(x_i) \ldots \omega_{2k-1} \sin(kx_i) + \omega_{2k} \cos(kx_i)$

$\phi(x)$ is:

$\phi(x) = [1, \sin(x), \cos(x), \sin(2x), \cos(2x) \ldots \sin(kx), \cos(kx)]$

② Express RSS in terms of $\{\phi(x_i); i = 1, 2, \ldots N\}$

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

$\hat{y} = \omega_0 + \omega_1 \sin(x_i) \ldots \omega_{2k} \cos(kx_i) = \omega^T \phi(x_i)$

$$\therefore RSS = \sum_{i=1}^{N} (y - \omega^T \phi(x_i))^2$$

In Matrix format

$Y = [y_1 \ldots y_n]^T$

$\phi = N \times (2k+1) \Rightarrow \phi(x_i)^T$

$$\therefore = \|Y - \phi\omega\|^2$$

Z

③ Parameters values of ω

$$\frac{\partial}{\partial \omega} \|Y - \phi\omega\|^2 = -2\phi^T(Y - \phi\omega) = 0$$

$$\Rightarrow -2\phi^T Y + 2\phi^T \phi\omega = 0$$

$$\Rightarrow -2\phi^T Y = -2\phi^T \phi\omega$$

$$\therefore \omega = (\phi^T \phi)^{-1} \phi^T Y$$

Z

8.3

① Value of K with minimum training error

$K = 10$

② Value of K with minimum validation error

$K = 5$

③ Why are they different.

As K increases after optimal point the model starts overfitting hence minimum at highest K whereas validation error reduces to the point then due to the model overfitting it starts increasing.

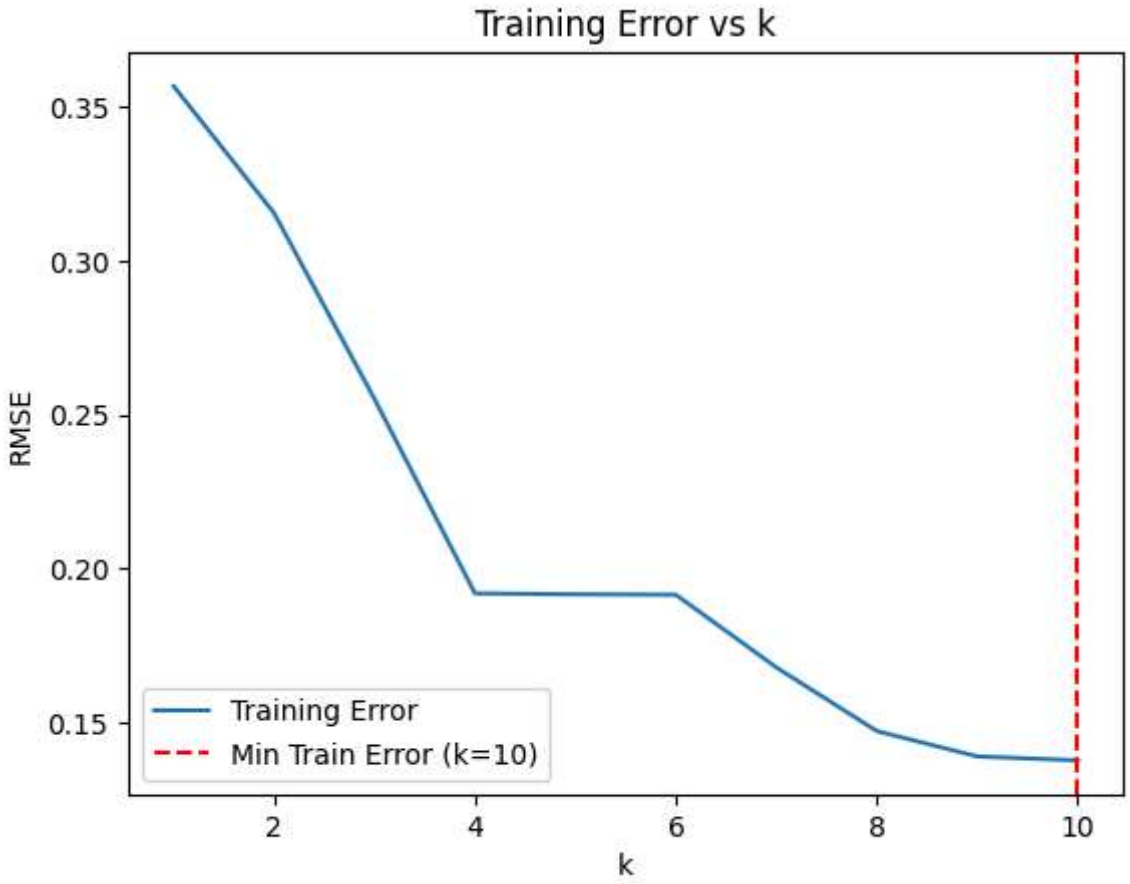
8. Non-linear Basis Functions

8.2 Implementation

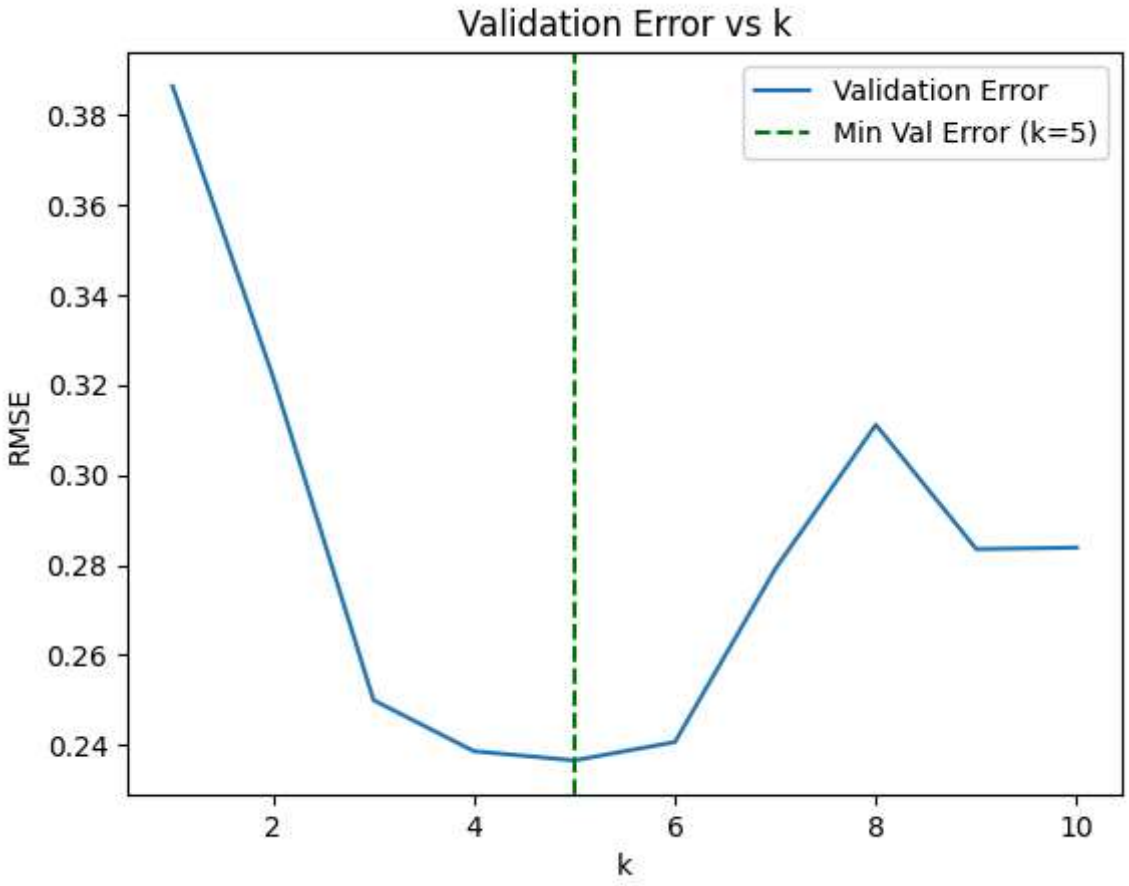
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from numpy.linalg import inv, pinv
from sklearn.model_selection import train_test_split


class SinusoidalRegressor:
    def __init__(self):
        self.k = None
        self.weights = None

    def phi(self, x):
        # The basis function for a general 2k
        phi_x = np.ones((x.shape[0], 2 * self.k + 1))
        for i in range(1, self.k + 1):
            phi_x[:, 2 * i - 1] = np.sin(i * x)
            phi_x[:, 2 * i] = np.cos(i * x)
        return phi_x

    def fit(self, X_train, Y_train, k):
        self.k = k
        # Construct the design matrix Phi for all data points in X_train
        Phi = self.phi(X_train)
        # Solve for the weights using the normal equation with a pseudo-inverse
        self.weights = pinv(Phi.T @ Phi) @ Phi.T @ Y_train

    def predict(self, X):
        # Check if the model is fitted
        if self.weights is None:
            raise ValueError("Model is not fitted yet.")
        # Apply the learned model
        Phi = self.phi(X)
        return Phi @ self.weights

    def rmse(self, X_val, Y_val):
        # Predict the values for X_val
        Y_pred = self.predict(X_val)
        # Calculate the RMSE
        return np.sqrt(np.mean((Y_val - Y_pred) ** 2))


np.random.seed(61)
csv_file = 'nonlinear-regression-data.csv'
data = pd.read_csv(csv_file)
x = np.array(data['X'])
y = np.array(data['Noisy_y'])
```

In [2]:
```
### Evaluation Part 0 ##################################################################

# Split the data
X_train, X_val, Y_train, Y_val = train_test_split(x, y, train_size=45, test_size=16


### Evaluation Part 1 and 2 ############################################################

# Initialize the model
model = SinusoidalRegressor()

train_errors = []
val_errors = []
k_values = range(1, 11)

for k in k_values:
    model.fit(X_train, Y_train, k)
    train_rmse = model.rmse(X_train, Y_train)
    val_rmse = model.rmse(X_val, Y_val)
    train_errors.append(train_rmse)
    val_errors.append(val_rmse)

# Find k values that give the minimum training and validation errors
k_min_train = k_values[train_errors.index(min(train_errors))]
k_min_val = k_values[val_errors.index(min(val_errors))]
```

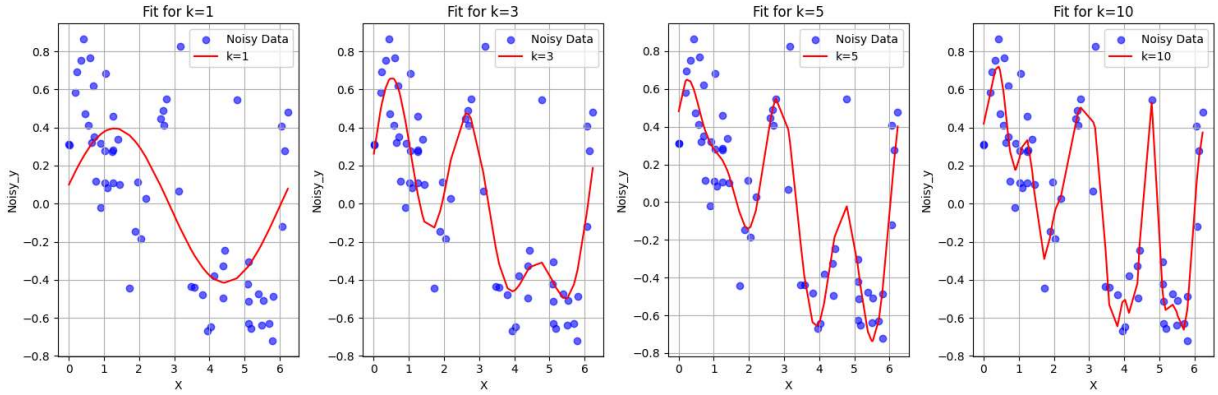
In [4]:
```
# Plotting the training error versus k
plt.figure()
plt.plot(k_values, train_errors, label='Training Error')
plt.axvline(x=k_min_train, color='r', linestyle='--', label=f'Min Train Error (k={k
plt.xlabel('k')
plt.ylabel('RMSE')
plt.title('Training Error vs k')
plt.legend()
plt.show()
```



In [5]:
```
# Plotting the validation error versus k
plt.figure()
plt.plot(k_values, val_errors, label='Validation Error')
plt.axvline(x=k_min_val, color='g', linestyle='--', label=f'Min Val Error (k={k_min
plt.xlabel('k')
plt.ylabel('RMSE')
plt.title('Validation Error vs k')
plt.legend()
plt.show()
```



In [6]:
```python
### Evaluation Part 4 ##################################################################
# You will create separate plots for each k you can use plt.subplots function
k_values = [1, 3, 5, 10]
fig, axes = plt.subplots(1, len(k_values), figsize=(15, 5))

for i, k in enumerate(k_values):
    model.fit(x, y, k)
    y_pred = model.predict(x)

    axes[i].scatter(x, y, label='Noisy Data', color='blue', alpha=0.6)
    axes[i].plot(x, y_pred, label=f'k={k}', color='red')
    axes[i].set_xlabel('X')
    axes[i].set_ylabel('Noisy_y')
    axes[i].set_title(f'Fit for k={k}')
    axes[i].legend()
    axes[i].grid(True)

plt.tight_layout()
plt.show()
```


Fit for k=1
Noisy Data
k=1
Fit for k=3
Noisy Data
k=3
Fit for k=5
Noisy Data
k=5
Fit for k=10
Noisy Data
k=10
Noisy_y
X

In [ ]: