Problem 2. SAHeart

```
import numpy as np

from scipy.optimize import minimize


data = np.genfromtxt("SAheart.data.txt", delimiter=",", skip_header=1, dtype=None,
encoding=None)


#Feature columns

feature_names = ['ldl', 'adiposity', 'typea', 'obesity', 'alcohol', 'age']

feature_indices = [2, 3, 6, 7, 8, 9]

label_index = 10  # chd


#Extract features and labels

X_all = np.array([[float(row[i]) for i in feature_indices] for row in data])

y_all = np.array([int(row[label_index]) for row in data])


#Labels are in -1, 1 for SVM

y_all = 2 * y_all - 1


#Last 16 examples

X = X_all[-16:]

y = y_all[-16:]


n_samples, n_features = X.shape

C = 2
```

```
#Linear kernel

K = X @ X.T

P = np.outer(y, y) * K


#Objective function

def objective(alpha):

   return 0.5 * alpha @ P @ alpha - np.sum(alpha)


#Gradient

def gradient(alpha):

   return P @ alpha - np.ones_like(alpha)


#Equality constraint

constraints = {'type': 'eq', 'fun': lambda a: np.dot(a, y), 'jac': lambda a: y}


#Bounds

bounds = [(0, C) for _ in range(n_samples)]


#Initial guess

alpha0 = np.zeros(n_samples)


# Solve

res = minimize(objective, alpha0, jac=gradient, constraints=constraints, bounds=bounds)

alpha_opt = res.x


print("\n===== PART A =====\n")
print("Dual variables (alpha):")

print(alpha_opt)


print("\n===== PART B =====\n")



w = np.sum((alpha_opt * y)[:, None] * X, axis=0)

print("Slope vector (w):")

print(w)


print("\n===== PART C =====\n")



support_indices = np.where((alpha_opt > 1e-5) & (alpha_opt < C - 1e-5))[0]

b_vals = y[support_indices] - X[support_indices] @ w

b = np.mean(b_vals)

print("Intercept (b):", b)


print("\n===== PART D =====\n")



#Decision function

decision_values = X @ w + b

predictions = np.sign(decision_values)
```

```
#Margin = 1
margins = y * decision_values

correct_outside = np.where((predictions == y) & (margins > 1))[0]
correct_on = np.where((predictions == y) & (np.isclose(margins, 1)))[0]
correct_inside = np.where((predictions == y) & (margins < 1))[0]
incorrect = np.where(predictions != y)[0]

print("Correctly classified outside margin:", correct_outside)
print("Correctly classified on margin:", correct_on)
print("Correctly classified inside margin:", correct_inside)
print("Incorrectly classified:", incorrect)

print("\n===== PART E =====\n")

print("Support vector indices:", np.where(alpha_opt > 1e-5)[0])

print("\n===== PART F =====\n")

distances = np.abs(decision_values) / np.linalg.norm(w)
margin = 1 / np.linalg.norm(w)
accuracy = np.mean(predictions == y)

print("Distances to hyperplane:", distances)
print("Margin size:", margin)
print("Accuracy:", accuracy)
print("\n===== PART G =====\n")

x_new = np.ones(X.shape[1])
pred_new = np.sign(x_new @ w + b)
print("Prediction for all features = 1:", int((pred_new + 1) // 2))  # Convert back to {0,1}
```

A. **Using only the Numpy and SciPy libraries, compute and print the dual variables that maximize the dual form.**

```
Dual variables (alpha):
[2.00000000e+00 2.68755621e-15 2.00000000e+00 0.00000000e+00
 1.39701219e+00 1.51133441e-16 0.00000000e+00 5.40288993e-02
 2.00000000e+00 5.05508552e-01 1.63210454e+00 2.00000000e+00
 2.00000000e+00 8.52255826e-01 3.73586128e-01 1.69542123e+00]
```

B. **Compute and print the slope parameter vector.**

```
Slope vector (w):
[-1.25678105  0.62812648 -0.08380015 -0.10500163  0.08864152  0.01781455]
```

C. **Compute and print the intercept parameter.**

```
Intercept (b): 3.8917063490377446
```

D. **Identify the indices of the examples that are (i) correctly classified outside the margin, (il) correctly classified on the boundary of the margin, (li) correctly classified inside the margin, (iv) incorrectly classified.**

I. Correctly classified outside margin: [1 3 5 6 7 9 13 14]
II. Correctly classified on margin: [4]
III. Correctly classified inside margin: [0 2 4 10 11 15]
IV. Incorrectly classified: [8 12]

E. **Identify the indices of the support vectors.**

```
Support vector indices: [ 0  2  4  7  8  9 10 11 12 13 14 15]
```

F. **Compute and print the distance of each data point from the decision hyperplane. Find the size of the margin. Find the accuracy of the classifier.**

```
Distances to hyperplane: [0.57588815 0.78597456 0.41865625 2.25374722 0.7070539
1.43173921
 3.08611095 0.70714174 0.31137606 0.70723145 0.70687444 0.12166486
 2.3988532  0.70719719 0.70709379 0.706786  ]
Margin size: 0.7070601228510428
Accuracy: 0.875
```

G. **Predict the label for all features being 1.**

```
Prediction for all features = 1: 1
```

Problem 3

```
import numpy as np

from sklearn.preprocessing import StandardScaler


#Load and transpose the data

xtrain = np.genfromtxt("khan.xtrain").T

xtest = np.genfromtxt("khan.xtest").T

ytrain = np.genfromtxt("khan.ytrain", dtype=int)

ytest = np.genfromtxt("khan.ytest", dtype=int)


print("\n===== PART A =====\n")


#Standardize training and test features

scaler = StandardScaler()

xtrain_std = scaler.fit_transform(xtrain)

xtest_std = scaler.transform(xtest)


#Check block

print("Training shape:", xtrain_std.shape)

print("Testing shape:", xtest_std.shape)

print("Training classes:", np.unique(ytrain))

print("Testing classes:", np.unique(ytest))



print("\n===== PART B =====\n")
```

```
from sklearn.svm import SVC
from sklearn.model_selection import LeaveOneOut, cross_val_score
import matplotlib.pyplot as plt

#Filter out -1 from test set
mask = ytest != -1
xtest_clean = xtest_std[mask]
ytest_clean = ytest[mask]

#Set up range of C values
C_values = np.logspace(-3, 3, 50)
test_accuracies = []
cv_accuracies = []

loo = LeaveOneOut()

for C in C_values:
    clf = SVC(kernel="linear", C=C)
    clf.fit(xtrain_std, ytrain)

    #Test accuracy
    test_acc = clf.score(xtest_clean, ytest_clean)
    test_accuracies.append(test_acc)

    # LOOCV accuracy
    cv_scores = cross_val_score(clf, xtrain_std, ytrain, cv=loo)
    cv_accuracies.append(np.mean(cv_scores))

#Plot results
plt.figure(figsize=(10, 5))
plt.plot(np.log10(C_values), test_accuracies, label="Test Accuracy", linewidth=2)
plt.plot(np.log10(C_values), cv_accuracies, label="LOOCV Accuracy", linewidth=2)
plt.xlabel("log10(C)")
plt.ylabel("Accuracy")
plt.title("Accuracy vs C for Linear SVM")
plt.legend()
plt.grid(True)
plt.show()

print("\n===== PART C =====\n")

#hoose any C (they all performed the same); we'll pick the middle one
optimal_C = C_values[len(C_values)//2]

#Linear kernel final model
clf_linear = SVC(kernel="linear", C=optimal_C)
clf_linear.fit(xtrain_std, ytrain)
train_acc_linear = clf_linear.score(xtrain_std, ytrain)
test_acc_linear = clf_linear.score(xtest_clean, ytest_clean)

#Predict on zero vector
zero_input = np.zeros((1, xtrain_std.shape[1]))
```

```
pred_linear = clf_linear.predict(zero_input)


print("Linear SVM")

print("Training accuracy:", train_acc_linear)

print("Test accuracy:", test_acc_linear)

print("Prediction for all-zero input:", pred_linear[0])


#RBF kernel

clf_rbf = SVC(kernel="rbf", C=optimal_C)

clf_rbf.fit(xtrain_std, ytrain)

train_acc_rbf = clf_rbf.score(xtrain_std, ytrain)

test_acc_rbf = clf_rbf.score(xtest_clean, ytest_clean)

pred_rbf = clf_rbf.predict(zero_input)


print("\nRBF SVM")

print("Training accuracy:", train_acc_rbf)

print("Test accuracy:", test_acc_rbf)

print("Prediction for all-zero input:", pred_rbf[0])
```

===== PART A =====

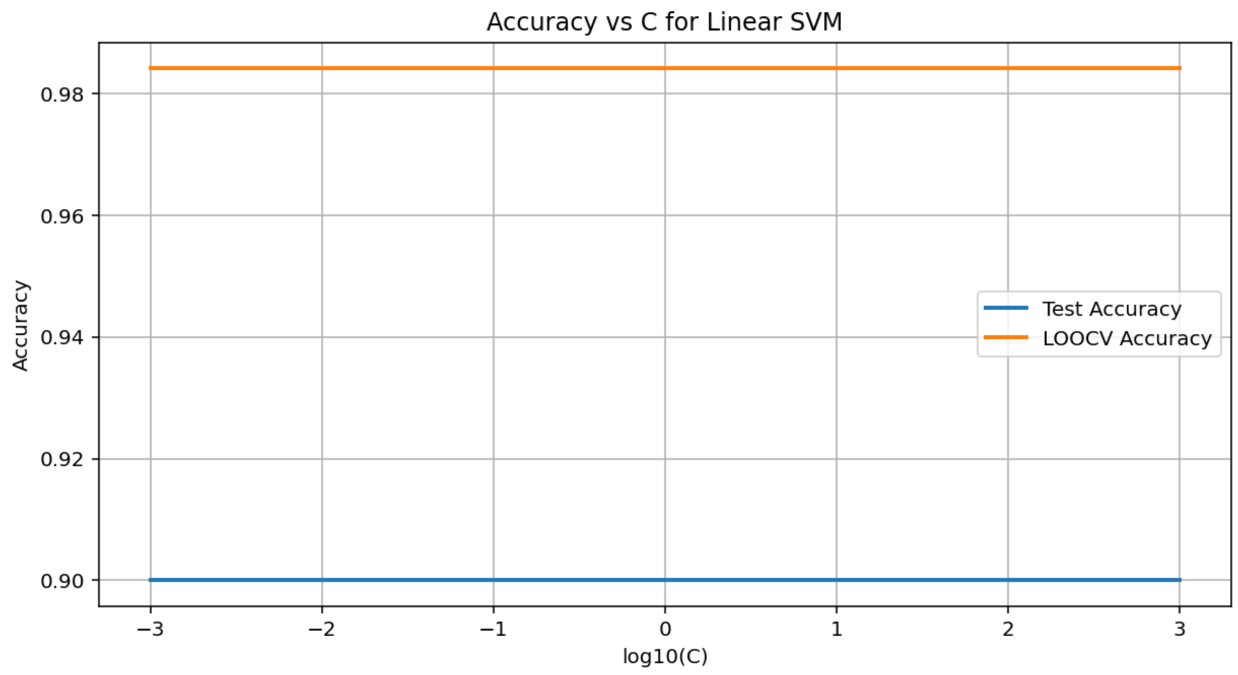
Training shape: (63, 2308)

Testing shape: (25, 2308)

Training classes: [1 2 3 4]

Testing classes: [-1 1 2 3 4]

===== PART B =====



As the value of C was varied across 50 different values, the test accuracy stayed at 90% and the LOOCV accuracy hovered around 98.4%. This shows that the model is very stable , as changing C didn't really affect performance. This suggests that the classes were already well-separated in the feature space, so adjusting the regularization parameter didn't make much of a difference.

===== PART C =====

Linear SVM

Training accuracy: 1.0

Test accuracy: 0.9

Prediction for all-zero input: 2

RBF SVM

Training accuracy: 1.0

Test accuracy: 0.5

Prediction for all-zero input: 3

When comparing the linear and RBF kernels, the linear one clearly worked better. The linear SVM got 100% accuracy on the training data and 90% on the test data. In contrast, the RBF kernel also got 100% training accuracy, but only managed 50% test accuracy. This shows that the RBF kernel was overfitting—it captured the training data perfectly but didn't generalize as well to new examples. Thus, the linear kernel is the better fit for this dataset.

Each model was also tested to see if what it would predict if we gave it an input vector of all zeros (just a row of zeroes). The linear SVM predicted class 2, and the RBF SVM predicted class 3. Although this input doesn't reflect a realistic case, the difference in predictions reinforces that the RBF model is more sensitive and potentially unstable when encountering unfamiliar data compared to the linear kernal.