###The professor mentioned for the Friday submission that we do not have to submit the code with the Scikit-Learn part

Problem 2.

```
# SVM with Dual Form using NumPy and SciPy only

import numpy as np

from scipy.optimize import minimize


X = np.genfromtxt("nci.data.csv", delimiter=",", dtype=float).T

X = np.nan_to_num(X, nan=0.0)


with open("nci.label.txt", "r") as f:

    labels = [line.strip() for line in f if line.strip() != ""]


# Fix for missing label in data if needed

if len(labels) < X.shape[0]:

    labels.append("UNKNOWN")


labels = np.array(labels)


# Filter only NSCLC (+1) and RENAL (-1)

mask = (labels == "NSCLC") | (labels == "RENAL")

X = X[mask]

y = np.where(labels[mask] == "NSCLC", 1, -1)


# Build the linear kernel
```

```
K = X @ X.T
P = np.outer(y, y) * K
n = X.shape[0]


def objective(alpha):
    """
    The SVM dual objective: (1/2)*alpha^T P alpha - sum(alpha).
    P includes y_i*y_j*K_{ij}.
    """
    return 0.5 * alpha @ P @ alpha - np.sum(alpha)


def svm_dual_constraint(alpha):
    """
    Enforces the SVM dual constraint:
    sum(alpha_i * y_i) = 0
    """
    return np.dot(alpha, y)

#Computation block


# Bounds and constraint
bounds = [(0, None)] * n
constraints = {"type": "eq", "fun": svm_dual_constraint}
alpha0 = np.zeros(n)


# Solve the optimization problem
```

```
res = minimize(objective, alpha0, bounds=bounds, constraints=constraints)
alphas = res.x

# Compute the slope vector w
w = ((alphas * y)[:, None] * X).sum(axis=0)

# Compute the intercept term b
support_indices = np.where(alphas > 1e-5)[0]
b = np.mean([y[i] - np.dot(w, X[i]) for i in support_indices])

# Print block
print("Dual variables (alphas):\n", alphas, "\n")
print("Slope vector (w):\n", w, "\n")
print("Intercept (b):", b, "\n")
print("Support vector indices:\n", support_indices, "\n")

distances = (X @ w + b) / np.linalg.norm(w)
print("Distances from the decision hyperplane:\n", distances, "\n")

predictions = np.sign(X @ w + b)
accuracy = np.mean(predictions == y)
print("Accuracy:", accuracy, "\n")

margin = 1.0 / np.linalg.norm(w)
print("Margin:", margin, "\n")
```

x_new = np.ones(X.shape[1])

pred_label = np.sign(np.dot(w, x_new) + b)

print("Prediction for all features = 1:", "NSCLC" if pred_label == 1 else "RENAL")

a. **compute and print the dual variables that maximize the dual form:**

Dual variables (alphas):

```
[3.70278657e-04 3.01033070e-04 4.46397864e-04 2.49846667e-04
 7.01584683e-05 2.55017977e-05 4.46448252e-04 1.50994606e-04
 1.14033099e-04 1.74384569e-04 1.68914487e-04 2.48013812e-04
 1.78839389e-04 2.33601040e-04 3.42191464e-04 3.73159695e-05
 1.15864802e-04 2.55018435e-05]
```

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

Slope vector (w):

```
[ 0.00000000e+00 -5.21888794e-04  7.06950790e-04 ...  4.28827108e-04
 -8.30583720e-05  5.08519002e-05]
```

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

Intercept (b): 0.443292031914553

d. **identify the indices of the support vectors**

Support vector indices:

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17]
```

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

Distances from the decision hyperplane:

[-16.28259853 16.5007701   16.58102246 -16.08756754 -16.56092696
 -18.09147119 -16.87225366 -17.2181869 -16.72237254 -17.296374
  15.76299432 -16.32106027 16.20170944 15.9985837   16.16599994
  16.29085067 16.17330577 21.77757519]

Accuracy: 1.0

f. Find the margin

Margin: 16.39058040244113

g. Predict the label for all features being 1.

Prediction for all features = 1: NSCLC

Problem 3

```
import numpy as np

from scipy.optimize import minimize



#skip_header=1 to skip the column header row

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


with open("SAheart.data", "r") as f:

   headers = f.readline().strip().split(",")


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

target_name = "chd"

feature_indices = [headers.index(name) for name in feature_names]

target_index = headers.index(target_name)


#only use the *first 20 rows* for this problem

X = np.array([[float(row[i]) for i in feature_indices] for row in data[:20]])

y = np.array([1 if int(row[target_index]) == 1 else -1 for row in data[:20]])


#Construct the linear kernel matrix
```

```python
K = X @ X.T

P = np.outer(y, y) * K  # shape (n,n)

n = X.shape[0]


def objective(alpha):

   """

   (a) SVM Dual Objective:

      0.5 * alpha^T (P) alpha - sum(alpha)

   where P = y*y^T .* (X X^T).

   """

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


def svm_dual_constraint(alpha):

   """

   Constrained to: sum(alpha_i * y_i) = 0

   to ensure a valid separating hyperplane for SVM.

   """

   return np.dot(alpha, y)


#Define bounds (alpha_i >= 0) and constraints

bounds = [(0, None) for _ in range(n)]

constraints = {"type": "eq", "fun": svm_dual_constraint}

alpha0 = np.zeros(n)


#run the optimizer to find alpha

res = minimize(objective, alpha0, bounds=bounds, constraints=constraints)
```

```
alphas = res.x

#Print block

print("(a) Dual variables (alphas):\n", alphas, "\n")

# (b) Compute & print the slope vector w = sum_i alpha_i y_i x_i
w = ((alphas * y)[:, None] * X).sum(axis=0)
print("(b) Slope vector (w):\n", w, "\n")

# (c) Compute & print the intercept b
support_indices = np.where(alphas > 1e-5)[0]
b = np.mean([y[i] - w @ X[i] for i in support_indices])
print("(c) Intercept (b):\n", b, "\n")

# (d) Print support vector indices
print("(d) Support vector indices: ", support_indices, "\n")

# (e) Distances & Accuracy
distances = (X @ w + b) / np.linalg.norm(w)
print("(e) Distances from hyperplane:\n", distances, "\n")

predictions = np.sign(X @ w + b)
accuracy = np.mean(predictions == y)
print("Accuracy:", accuracy, "\n")
```

```
# (f) Margin = 1 / ||w||
margin = 1.0 / np.linalg.norm(w)
print("(f) Margin:", margin, "\n")

# (g) Predict label when x=1 for all features
x_new = np.ones(X.shape[1])
pred_label = np.sign(w @ x_new + b)
print("(g) Prediction for (all features=1):",
    "CHD" if pred_label == 1 else "No CHD")
```

a. **compute and print the dual variables that maximize the dual form:**

```
(a) Dual variables (alphas):
 [1.14414968e-03 1.79990124e-13 3.67738988e-13 8.85818122e-01
 3.11377307e-14 1.27951034e+00 3.34267783e-13 1.80592981e-13
 0.00000000e+00 8.85976642e-02 0.00000000e+00 1.12261754e+00
 0.00000000e+00 0.00000000e+00 7.32206911e-01 2.15305828e-13
 8.64602208e-02 1.43375356e-13 0.00000000e+00 6.78261849e-13]
```

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

```
(b) Slope vector (w):
 [-0.09768246  0.76534166  0.2597387  -1.83248668  0.02896907  0.41584209]
```

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

```
(c) Intercept (b):
 -6.918286936334213
```

d. **identify the indices of the support vectors**

(d) Support vector indices: [0 3 5 9 11 14 16]

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

(e) Distances from hyperplane:

[0.49505731 1.06775734 -1.56545379 0.49245178 1.9512953 -0.48659341

-0.86984256 0.91593522 -6.41665269 0.48610055 6.65894333 0.48269468

-8.07746514 -8.90511454 -0.49378332 -1.63742713 -0.48765922 1.00522143

3.8602865 0.52646971]

Accuracy: 1.0

f. **Find the margin**

(f) Margin: 0.4882683751314705

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

(g) Prediction for (all features=1): No CHD