```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 30 01:19:57 2020

@author: melodychen
"""

import numpy as np
import matplotlib.pyplot as plt
import csv
import cvxpy as cp

# global variables
x_train = []
y_train = []
x_1_1 = []
x_1_0 = []
x_2_1 = []
x_2_0 = []


def load_data():
    global x_1_1, x_1_0, x_2_1, x_2_0
    x_1_1 = []
    x_1_0 = []
    x_2_1 = []
    x_2_0 = []
    global x_train, y_train
    x_train = []
    y_train = []
    with open("data.csv") as csvfile:
        readCSV = csv.reader(csvfile, delimiter=',')
        for row in readCSV:
            if int(row[2]) == 1:
                x_1_1.append(float(row[0]))
                x_2_1.append(float(row[1]))
            else:
                x_1_0.append(float(row[0]))
                x_2_0.append(float(row[1]))
```

```
41              x_train.append([float(row[0]), float(row[1
    ])])
42              y_train.append(float(row[2]))
43      # plot all points
44      plt.scatter(x_1_1, x_2_1, label='Label: 1', color=
    'red')
45      plt.scatter(x_1_0, x_2_0, label='Label: -1', color
    ='orange')
46      plt.xlabel('x1')
47      plt.ylabel('x2')
48      plt.title("SVM Plot for Data.csv")
49      plt.axis([min(x_1_0 + x_1_1) - 1, max(x_1_0 +
    x_1_1) + 1, min(x_2_1 + x_2_0) - 1, max(x_2_1 + x_2_0
    ) + 1])
50
51
52  def primal_problem():
53      # load data
54      x = np.zeros(shape=(len(x_train), 2))
55      y = np.zeros(shape=(len(x_train), 1))
56      # load into numpy array
57      for index, row in enumerate(x_train):
58          x[index][0] = row[0]
59          x[index][1] = row[1]
60          y[index] = y_train[index]
61      # variables we're using to minimize
62      w = cp.Variable(2)
63      b = cp.Variable(1)
64      # function we're trying to minimize
65      cost = cp.sum_squares(w)
66      # constraints for minimization
67      constraints = []
68      for index, row in enumerate(x):
69          constraints.append(y[index] * (w.T @ row + b
    ) >= 1)
70      # use cvxpy to do minimization
71      prob = cp.Problem(cp.Minimize(cost), constraints)
72      result = prob.solve()
73      # final values
74      print("w vector: "+str(w.value))
```

```
75      print("b: "+str(b.value))
76      # plot line perpendicular to vector w, decision
    boundary
77      x_line = np.linspace(min(x_1_0 + x_1_1)-1, max(
    x_1_0 + x_1_1)+1, 100)
78      # equation of line, we know that w1*x1 + w2*x2 +
    b = 0, x2 = (-w1*x1)/w2 - b/w2
79      y_line = (-(float(w.value[0])/float(w.value[1]))*
    x_line-(float(b.value)/float(w.value[1])))
80      plt.plot(x_line, y_line, color='blue', label='
    Decision Boundary')
81
82
83  def dual_problem():
84      # load data
85      x = np.zeros(shape=(len(x_train), 2))
86      y = np.zeros(shape=(len(x_train), 1))
87      for index, row in enumerate(x_train):
88          x[index][0] = row[0]
89          x[index][1] = row[1]
90          y[index] = y_train[index]
91      # variable used to maximize dual
92      alpha = cp.Variable(len(x_train))
93      # P matrix that represents part of latter part of
     W(a)
94      p = np.zeros(shape=(len(x_train), len(x_train)))
95      # Fill up P matrix with y_i*y_j*x_i^T*x_j
96      for r in range(len(x)):
97          for c in range(len(x)):
98              p[r][c] = y[r]*y[c]*x[r].transpose().dot(
    x[c])
99      # slight adjustment to P
100     p = p + 1e-13 * np.eye(31)
101     # function we're trying to maximize
102     cost = sum(alpha) - 0.5 * cp.quad_form(alpha, p)
103     # our constraints
104     constraints = []
105     for a in alpha:
106         constraints.append(0 <= a)
107     constraints.append(sum(alpha * y) == 0)
```

```
108     # using cvxpy to solve maximization problem
109     prob = cp.Problem(cp.Maximize(cost), constraints)
110     result = prob.solve()
111     print("Original Alphas: ")
112     print(alpha.value)
113     # we want to make very small values zero
114     alpha_float = []
115     non_zero_alpha = []
116     for index, num in enumerate(alpha.value):
117         if float(num) < 1e-9:
118             alpha_float.append(0)
119         else:
120             alpha_float.append(float(num))
121             non_zero_alpha.append((index, float(num
    )))
122     print("Cleaned up version of Alphas: ")
123     print(alpha_float)
124     print(non_zero_alpha)
125     # want to highlight these points
126     x_1_highlight = []
127     x_2_highlight = []
128     for tup in non_zero_alpha:
129         x_1_highlight.append(x[tup[0]][0])
130         x_2_highlight.append(x[tup[0]][1])
131     # highlights support vector in plot
132     plt.scatter(x_1_highlight, x_2_highlight, color='
    purple', label='Label: Support Vector')
133
134
135 if __name__ == "__main__":
136     load_data()  # part a
137     primal_problem()  # part b
138     dual_problem()  # part c
139     # show legend
140     plt.legend(loc='upper right')
141     # plot the graph
142     plt.show()
143
```