Universidade Tecnológica Federal do Paraná
Campus Cornélio Procópio
Departamento Acadêmico de Elétrica

Introdução ao Machine Learning usando Python – Lista 9

1 Objetivo

Implementar o código do algoritmo **Support Vector Machine**. Dica:
`http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote09.html`

2 Support Vector Machine (SVM)

O algoritmo *Support Vector Machine (SVM)* é um classificador linear binário. Sua tarefa é idêntica a tarefa do *Perceptron*. A diferença principal é que Perceptron gera como resultado um **hiperplano qualquer** e o SVM gera o **hiperplano ótimo**. O SVM é bastante usado em aplicações industriais de Machine Learning.

Problema: É conhecido um conjunto de dados

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

no qual cada $x_i \in \mathbb{R}^d$ é um vetor (dimensão d) e cada $y_i \in Y$ é valor $+1$ ou -1.

- SVM funciona somente se a hipótese a seguir é válida: existe um hiperplano que separa os dados linearmente.
- SVM funciona somente para classificação binária, ou seja, os dados só podem ter rótulo $+1$ ou -1.

Considere a imagem da Figura 1 e suponha que pontos em "verde" representam -1 e pontos em "vermelho" representam $+1$. No exemplo da Figura 1, os pontos denominados *support vectors* são dois

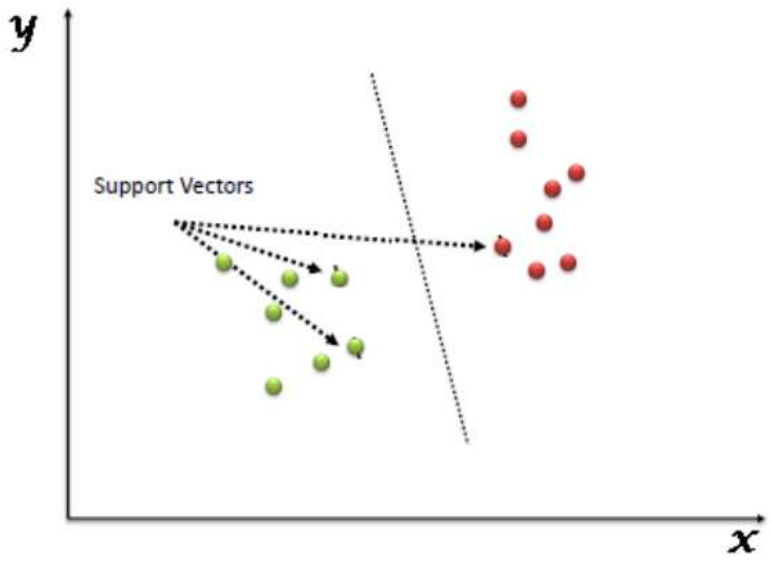

Figura 1: Exemplo de SVM

pontos para verde e um único ponto para vermelho. Note que a distância entre os pontos *support vectors* e o hiperplano (tracejado) são idênticos: o hiperplano faz a simetria perfeita, realizando a equi-distância entre os pontos *support vectors*.

2.1 Problema de otimização

No problema de SVM, desejamos encontrar os vetores $w \in \mathbb{R}^d$, $b \in \mathbb{R}^d$, de modo a calcular a solução do problema de otimização

$$\begin{aligned} &\min_{w,b} w^T w \\ &\text{sujeito a} \quad y_i(w^T x_i + b) \geq 1, \quad \forall i = 1, \ldots, n. \end{aligned} \tag{1}$$

Note acima que cada $x_1, x_2, \ldots, x_i, \ldots x_n$ tem dimensão d e eles são dados, e que cada $y_1, \ldots, y_i, \ldots y_n$ é $+1$ ou -1 (valores y_i também são dados). As variáveis são w e b.

3 Otimização de função não-linear sujeito a restrições não-lineares em Python

Python possui biblioteca chamada `scipy.optimize`, e essa biblioteca é bastante eficiente para resolver problema de otimização de função não-linear sujeito a restrições não-lineares na forma

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &\text{sujeito à} \quad h(x) = 0, \quad g(x) \geq 0. \end{aligned}$$

Acima, representamos x como sendo a variável a ser determinada.

`https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html`

3.1 Tarefa 1

Suponha que desejamos obter a solução do problema de otimização abaixo:

$$\begin{aligned} &\min\ x_1 x_4 (x_1 + x_2 + x_3) + x_3 \\ &\text{sujeito à} \quad x_1 x_2 x_3 x_4 \geq 25 \\ &\qquad\qquad x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40 \\ &\qquad\qquad 1 \leq x_1, x_2, x_3, x_4 \leq 5 \\ &\qquad\qquad x_0 = (1, 5, 5, 1) \end{aligned}$$

Podemos usar a biblioteca `scipy.optimize` e o código abaixo para resolver o problema (código-fonte disponível nos links a seguir);

`apmonitor.com/pdc/index.php/Main/NonlinearProgramming`

`stackoverflow.com/questions/21765794/python-constrained-non-linear-optimization`

```
1
2
3  import numpy as np
4  from scipy.optimize import minimize
5
6  def objective(x):
7      return x[0]*x[3]*(x[0]+x[1]+x[2])+x[2]
8
9  def constraint1(x):
10     return x[0]*x[1]*x[2]*x[3]-25.0
11
12 def constraint2(x):
13     sum_eq = 40.0
14     for i in range(4):
15         sum_eq = sum_eq - x[i]**2
16     return sum_eq
```

```
17
18 # initial guesses
19 n = 4
20 x0 = np.zeros(n)
21 x0[0] = 1.0
22 x0[1] = 5.0
23 x0[2] = 5.0
24 x0[3] = 1.0
25
26 # show initial objective
27 print('Initial SSE Objective: ' + str(objective(x0)))
28
29 # optimize
30 b = (1.0,5.0)
31 bnds = (b, b, b, b)
32 con1 = {'type': 'ineq', 'fun': constraint1}
33 con2 = {'type': 'eq', 'fun': constraint2}
34 cons = ([con1,con2])
35 solution = minimize(objective,x0,method='SLSQP',\
36                     bounds=bnds,constraints=cons)
37 x = solution.x
38
39 # show final objective
40 print('Final SSE Objective: ' + str(objective(x)))
41
42 # print solution
43 print('Solution')
44 print('x1 = ' + str(x[0]))
45 print('x2 = ' + str(x[1]))
46 print('x3 = ' + str(x[2]))
47 print('x4 = ' + str(x[3]))
```

3.2 Tarefa 2

2. Implemente o algoritmo SVM para encontrar o hiperplano que classifica as folhas em *setosa* e *versicolor*. Use o código abaixo a construa o código indicado como `otimizacao` (use ideias da Tarefa 1).

```
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from sklearn.linear_model import Perceptron
5
6  url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
7
8  # Assign colum names to the dataset
9  names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
10
11 # Read dataset to pandas dataframe
12 df = pd.read_csv(url, names=names)
13
14 vecSetosa = df.iloc[0:50,0:2].values.copy()
15 vecVersicolor = df.iloc[50:100,0:2].values.copy()
16
17 X =  df.iloc[0:100,0:2].values.copy()
18 y = np.append( -1*np.ones(50), np.ones(50) )
19
20 ############  otimizacao ############
21
22 # construa aqui teu codigo otimizacao
```

Listing 1: Comando Python para Tarefa 2

`https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html`