Classification as a QUBO problem

Nurali Bibolat

Introduction

+ Representing the search for optimal parameters (w, b) to construct the bestfor the decision boundary

$$(w^T x + b = 0\ )$$

for classifying the dataset, as a Quadratic Unconstrained Binary Optimization Problem

+ Use Grover’s Adaptive Search with Phase Encoding algorithm to find the bit-string used for figuring out the optimal values

Classification

- m training data points $x^{(i)}$

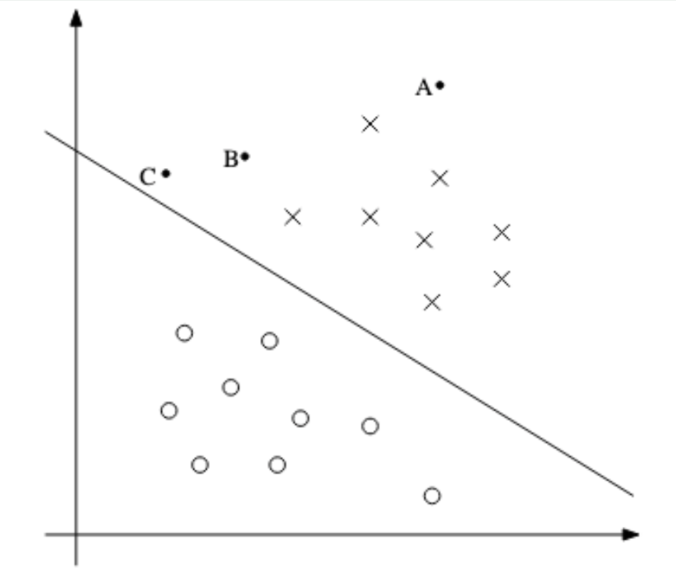
$$\begin{bmatrix} x_1^i \\ x_2^i \\ x_3^i \\ \vdots \\ x_n^i \end{bmatrix}$$

- associated output data points $y^{(i)}$
  - either -1 or 1

- A classifier function, $g(w^T x + b)$ maps input data points to the output data points, by finding the optimal weights and constant

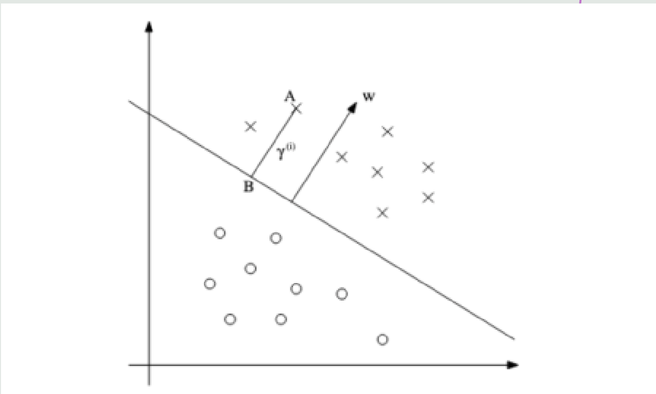
$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix}, \text{ and optimal } b \text{ constant.}$$

- Once the optimal weights and constants are found, a decision boundary is formed: $w^T x + b = 0$



- $w^T x + b >> 0$
  - confident that the $g(w^T x + b) = 1$

- Functional margin: $\hat{\gamma} = y^{(i)}(w^T x + b)$
- Geometric margin: $\gamma^{(i)} = \hat{\gamma}^{(i)} / ||w||$



Optimization problem

- Largest distance between data points and decision boundary
  - Further == more confident in prediction

$$\begin{aligned} \max_{\gamma,w,b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \ldots, m \\ & ||w|| = 1. \end{aligned}$$

$$\begin{aligned} \max_{\gamma,w,b} \quad & \frac{\hat{\gamma}}{||w||} \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \hat{\gamma}, \quad i = 1, \ldots, m \end{aligned}$$

$\hat{\gamma} = 1.$ (scale w, b)

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

Derivation from:
https://see.stanford.edu/materials/aimlcs229/cs229-notes3.pdf

Lagrange multipliers + KKT

Given the following optimization problem:

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & h_i(w) = 0, \quad i = 1, \ldots, l. \end{aligned}$$

$$\mathcal{L}(w, \beta) = f(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

Karush-Kuhn-Tucker (KKT) conditions, which are as follows:

$$\begin{aligned} \frac{\partial}{\partial w_i}\mathcal{L}(w^*, \alpha^*, \beta^*) &= 0, \quad i = 1, \ldots, n \\ \frac{\partial}{\partial \beta_i}\mathcal{L}(w^*, \alpha^*, \beta^*) &= 0, \quad i = 1, \ldots, l \\ \alpha_i^* g_i(w^*) &= 0, \quad i = 1, \ldots, k \\ g_i(w^*) &\leq 0, \quad i = 1, \ldots, k \\ \alpha^* &\geq 0, \quad i = 1, \ldots, k \end{aligned}$$

Maximize by varying alpha

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}.$$

$$\begin{aligned} \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

Minimize:

$$\sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}$$

$min_\alpha \sum_{i,j=1}^{m} \alpha_i \alpha_j q_{i,j}$

this problem is an NP-hard problem that takes the form of

$$x = \min_{x \in X} \sum_{i,j=0}^{2^n - 1} q_{ij} x_i x_j$$

Converting to a QUBO problem

$$min_\alpha \sum_{i,j=1}^{m} \alpha_i \alpha_j q_{i,j}$$

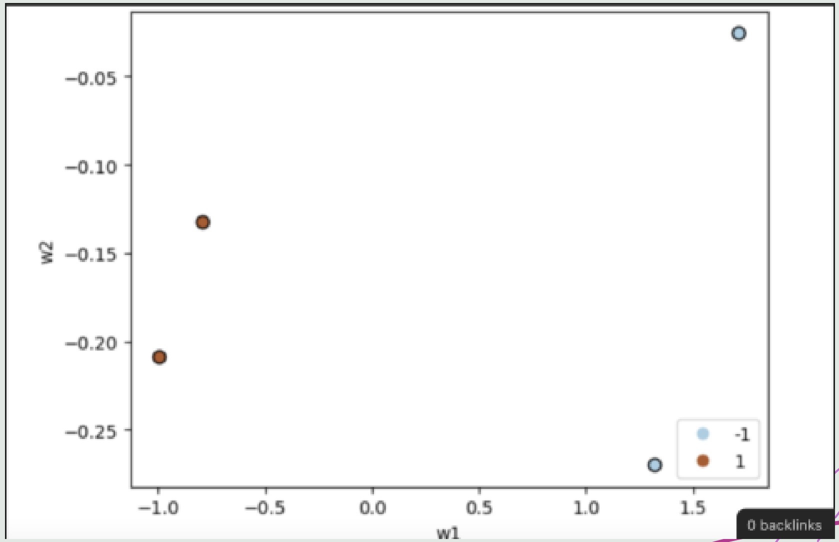
$$x = \min_{x \in X} \sum_{i,j=0}^{2^n-1} q_{ij} x_i x_j$$

$$\alpha_i = \sum_{k=1}^{n} x_{i,k} 2^{k-1}$$
$$\alpha_j = \sum_{l=1}^{n} x_{j,l} 2^{l-1}$$

$$min_x f(x)$$
$$f(x) = \sum_{i,j=1}^{m} (\sum_{k=1}^{n} x_{i,k} 2^{k-1}) * (\sum_{l=1}^{n} x_{j,l} 2^{l-1}) * q_{i,j}$$

Example

	flavanoids	malic_acid	target
0	1.321944	-0.270148	-1
1	1.714484	-0.025601	-1
2	-0.992685	-0.209012	1
3	-0.789647	-0.132590	1



- 12 + 13 = 25 qubits
- Transpile(qc, backend) doesn't work for more than 29
- See whether a sensible boundary is constructed
    - (not horizontal)

Example

- Use the Grover's Adaptive Search with Phase Encoding algorithm

f(x) =

```
'x_ik(1, 1, 0)*x_jl(1, 1, 0) *1.820516541074912 +  2*x_ik(1, 1, 0)*x_jl(1, 2, 1) *1.82
0516541074912 +  4*x_ik(1, 1, 0)*x_jl(1, 3, 2) *1.820516541074912 +  x_ik(1, 1, 0)*x_j
l(2, 1, 3) *2.273367695306561 +  2*x_ik(1, 1, 0)*x_jl(2, 2, 4) *2.273367695306561 +  4
*x_ik(1, 1, 0)*x_jl(2, 3, 5) *2.273367695306561 +  x_ik(1, 1, 0)*x_jl(3, 1, 6) *1.2558
098032447949 +  2*x_ik(1, 1, 0)*x_jl(3, 2, 7) *1.2558098032447949 +  4*x_ik(1, 1, 0)*x
_jl(3, 3, 8) *1.2558098032447949 +  x_ik(1, 1, 0)*x_jl(4, 1, 9) *1.0080504032334583 +
2*x_ik(1, 1, 0)*x_jl(4, 2, 10) *1.0080504032334583 +  4*x_ik(1, 1, 0)*x_jl(4, 3, 11) *
1.0080504032334583 +  2*x_ik(1, 2, 1)*x_jl(1, 1, 0) *1.820516541074912 +  4*x_ik(1, 2,
1)*x_jl(1, 2, 1) *1.820516541074912 +  8*x_ik(1, 2, 1)*x_jl(1, 3, 2) *1.82051654107491
2 +  2*x_ik(1, 2, 1)*x_jl(2, 1, 3) *2.273367695306561 +  4*x_ik(1, 2, 1)*x_jl(2, 2, 4)
*2.273367695306561 +  8*x_ik(1, 2, 1)*x_jl(2, 3, 5) *2.273367695306561 +  2*x_ik(1, 2,
```

```
def Grover1(xlist,couplings,y,r):

  n = 12
  m = 13
  qi=QuantumRegister(n,name='input')
  qo=QuantumRegister(m,name='output')
  c=ClassicalRegister(m)
  d=ClassicalRegister(n)
  qc=QuantumCircuit(qi,qo,c,d)

  qc.h(qi)


  print('y_init=',y)
  coeffs=couplings.copy()
  coeffs.append(y)


  AOperator(qc,qi,qo,xlist,coeffs)
  qc.barrier()



  for i in range(r):

    qc.z(qo[m-1])
    qc.barrier()


    AOpdagger(qc,qi,qo,xlist,coeffs)

    qc.barrier()
    qc.h(qi)
    qc.x(qi)
    qc.mcp(np.pi,qi[1:],qi[0])
    qc.x(qi)
    qc.h(qi)
    qc.barrier()

    AOperator(qc,qi,qo,xlist,coeffs)

  qc.barrier()

  qc.measure(qo,c)
  qc.measure(qi,d)

  backend = Aer.get_backend('qasm_simulator')
  circ = transpile(qc, backend)

  counts = backend.run(circ, shots=1024).result().get_counts(circ)

  return counts, qc
```

```
maxiter=1000
score=764
y=-score
bestx=0
r=1
cutoff = -4

for i in range(maxiter):
  print('r=',r)
  counts, q = Grover1(xlist,couplings,y,r)

  x = max(counts, key=lambda key: counts[key])
  print('x = ',x)


  ##########f x#################


  expandedd = str(expanded)

  for key, value in dctl.items():
    expandedd = expandedd.replace(f"{key}", f"{x[value]}")
  fx = eval(expandedd)


  ##########f x#################


  print('f(x) =',fx)
  xx = [int(str(x[:3]),2), int(str(x[3:6]),2), int(str(x[6:9]),2), int(str(x[9:12]),2)]
  print((-1 * xx[0]) + (-1 * xx[1]) + xx[2] + xx[3])
  if fx<score:    #Is the function value better than the best so far?
    r=1           #If so, update the variables
    y+=1
    score=fx
    bestx=x
    if score<cutoff:
      break
  else:
    if r<3:       #If the value isn't better, try again with a new number of iterations
      r+=1
    else:         #Or just move on
      r=1
      y+=1
print('best x=',bestx,' with a score of ',score)
```

```
r= 1
y_init= -764
x =  011111000101 0000011001110
```

Example

+ Given amount of training data points (m) = 4, n = 3 qubits per each alpha (alpha can take max value of 7)

```
y_init= -764
x =  010010001011 0000001001000
f(x) = 219.57387865924778
0
r= 2
y_init= -764
x =  111010011011 0000011100000
f(x) = 546.4754837280741
-3
r= 3
y_init= -764
x =  010110110111 0000100001010
f(x) = 266.33934231988127
5
r= 1
y_init= -763
x =  100000010010 0000000011101
f(x) = 24.049554652731366
0
r= 1
```

- for each y_init, the algorithm takes about 3-3.5 minutes to output the x, f(x), $\sum_{i=1}^{m} \alpha_i y_i,$, and r
- Calculated x and f(x) from -778 to -708
- Chose x = 100000010010 for which f(x) is very low relative to all other f(x), and satisfies other conditions

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 2 \\ 2 \end{bmatrix}$$

Results

$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$

```
W = np.zeros(2)
alpha = [4, 0, 2, 2]
for i in range(len(w)):
    #print(v(i))
    #print(np.array(v(i)[0]))
    W += np.array(v(i)[0]) * y(i) * alpha[i]
W
```

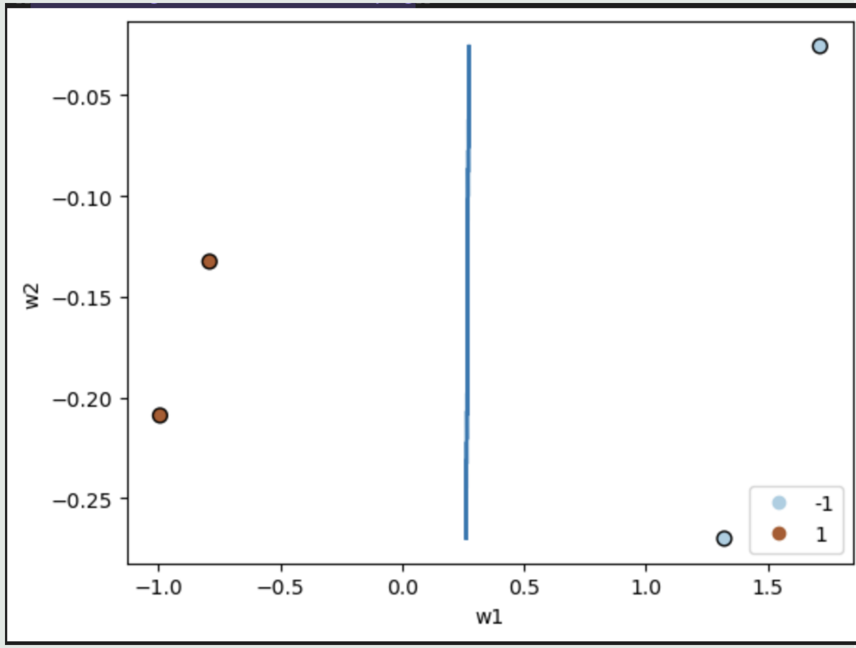
```
array([-8.85244 ,  0.397388])
```

$b = -\frac{(max_{i:y^{(i)}=-1} w^T x^{(i)}) + (min_{i:y^{(i)}=1} w^T x^{(i)})}{2}$

```
maxx = max([W.T @ np.array(v(i)[0]) for i in range(len(w)) if y(i) == -1])
minn = min([W.T @ np.array(v(i)[0]) for i in range(len(w)) if y(i) == 1])

#print(maxx, minn)
b = -(maxx + minn)/2
print(b)
```

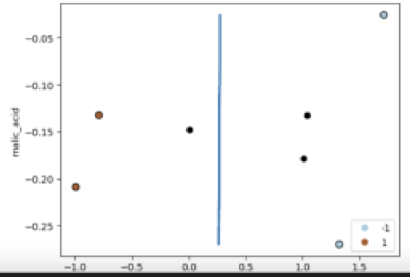
```
2.4360852515120004
```




```
In [146]: testw.iloc[1:4]
Out[146]:
```

	flavanoids	malic_acid	target	predicted
1	1.010620	-0.178443	-1	-1
2	1.037691	-0.132590	-1	-1
3	0.008967	-0.147875	-1	1

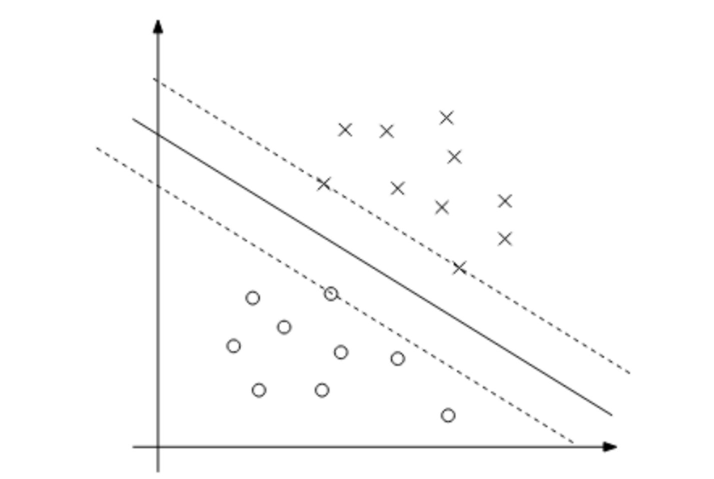
```
In [143]: plt.figure(1)
plot = plt.scatter(w['flavanoids'], w['malic_acid'], c=w['target'], edgecolor='k', s=50
plt.xlabel('flavanoids')
plt.ylabel('malic_acid')
plt.plot(w1, w2)
plt.scatter(testw1[1:4], testw2[1:4], color = 'black')
plt.legend(handles=plot.legend_elements()[0], labels=['-1', '1'], loc='lower right')
plt.show()
```



- the performance is quite poor (2/3 success) due to the following limitations of the implementation

Errors

- Algorithm with lagrange multipliers works best if we use data points with functional margins = 1 during training
  - At the optimal solution, KKT duality complementary conditions states that
  → $\alpha_i > 0$ occurs only for training data points that have functional margin = 1
  - The implementation used data points with functional margins > 0, due to lack of training data points



- The three data points used ($\hat{\gamma} = y^{(i)}(w^T x + b)$ = 1) are the support vectors
  - used to create the optimal decision boundary for data points near it
  - QUBO Support Vector implementation would utilize only these datapoints when constructing f(x)

- Need more training data points
- Only applicable for linearly separable datasets (no intersections)
  - Non-separable →reformulate the optimization problem again → new algorithms + Kernels → Support Vector Machines learning in high-dimensional feature space → ???

Source:
https://see.stanford.edu/materials/aimlcs229/cs229-notes3.pdf
https://see.stanford.edu/Course/CS229/37

Thank you!