Hillary vs. Bernie

[1]:
```python
import cvxpy as cp
import numpy as np
import scipy
mat = scipy.io.loadmat('Hillary_vs_Bernie.mat')
X = mat['features_train']
y = mat['labels_train']
m,n = X.shape
Y = np.zeros((m,m),float)
for i in range(m):
    Y[i][i] = y[i][0]
```

Fitting the model for $\gamma \in \{0.1, 1, 10\}$

The optimization problem

$$\begin{aligned} \min_{a,b,\eta} \; & ||a|| + \gamma \, ||\eta||_1 \\ \text{s.t. } & y_i(a^\top x_i - b) \geq 1 - \eta_i \qquad \forall i = 1, \cdots, m \\ & \eta \geq 0 \end{aligned}$$

to build a linear classifier. Corresponding to the three cases `gamma1 = 0.1`, `gamma2 = 1`, `gamma3 = 10`, the optimal solutions are labelled as `(a1,b1,eta1)`, `(a2,b2,eta2)`, `(a3,b3,eta3)` respectively.

Here's how we deal with the linear separator on the given data. I formed a matrix $(Y_{\text{train}} =)Y = diag(y_1, \cdots, y_m)$. The rows of $(X_{\text{train}} =)X$ are the vectors $x_i^\top$. So $Xa - b\mathbf{1}$ already gives the evaluation of the linear form on these data points $\{x_i\}$. We want to weigh each $x_i^\top a - b$ with y_i: this is achieved by taking $Y(Xa - b\mathbf{1})$ which gives a vector with i^{th} entry being $y_i(x_i^\top a - b)$.

[2]:
```python
gamma1 = 0.1
a1 = cp.Variable(n, 'a1')
b1 = cp.Variable(1,'b1')
eta1 = cp.Variable(m, 'eta1')
obj1 = cp.norm(a1) + gamma1 * (cp.norm(eta1,1))
cons1 = [Y@(X@a1-b1) + eta1 >= 1, eta1 >= 0]
problem1 = cp.Problem(cp.Minimize(obj1), cons1)
print(problem1.solve(verbose = True, solver = cp.ECOS))
print("\nOptimal a: ", a1.value, "\nOptimal b:", b1.value)
```

```
===============================================================================
                                     CVXPY
```

```
                                    v1.4.2
===============================================================================
(CVXPY) Mar 20 09:52:49 AM: Your problem has 181 variables, 2 constraints, and 0
parameters.
(CVXPY) Mar 20 09:52:49 AM: It is compliant with the following grammars: DCP,
DQCP
(CVXPY) Mar 20 09:52:49 AM: (If you need to solve this problem multiple times,
but with different data, consider using parameters.)
(CVXPY) Mar 20 09:52:49 AM: CVXPY will first compile your problem; then, it will
invoke a numerical solver to obtain a solution.
(CVXPY) Mar 20 09:52:49 AM: Your problem is compiled with the CPP
canonicalization backend.
-------------------------------------------------------------------------------
                                  Compilation
-------------------------------------------------------------------------------
(CVXPY) Mar 20 09:52:49 AM: Compiling problem (target solver=ECOS).
(CVXPY) Mar 20 09:52:49 AM: Reduction chain: Dcp2Cone -> CvxAttr2Constr ->
ConeMatrixStuffing -> ECOS
(CVXPY) Mar 20 09:52:49 AM: Applying reduction Dcp2Cone
(CVXPY) Mar 20 09:52:49 AM: Applying reduction CvxAttr2Constr
(CVXPY) Mar 20 09:52:49 AM: Applying reduction ConeMatrixStuffing
(CVXPY) Mar 20 09:52:49 AM: Applying reduction ECOS
(CVXPY) Mar 20 09:52:49 AM: Finished problem compilation (took 1.305e-02
seconds).
-------------------------------------------------------------------------------
                                Numerical solver
-------------------------------------------------------------------------------
(CVXPY) Mar 20 09:52:49 AM: Invoking solver ECOS  to obtain a solution.
-------------------------------------------------------------------------------
                                    Summary
-------------------------------------------------------------------------------
(CVXPY) Mar 20 09:52:49 AM: Problem status: optimal
(CVXPY) Mar 20 09:52:49 AM: Optimal value: 1.057e+01
(CVXPY) Mar 20 09:52:49 AM: Compilation took 1.305e-02 seconds
(CVXPY) Mar 20 09:52:49 AM: Solver (including time spent in interface) took
4.034e-03 seconds
10.57269665702442

Optimal a:  [ 0.14105247  0.18277618 -0.73224986 -0.10977297  0.38083898]
Optimal b: [-3.14700164]
```

[3]:

```
gamma2 = 1
a2 = cp.Variable(n, 'a2')
b2 = cp.Variable(1,'b2')
eta2 = cp.Variable(m, 'eta2')
obj2 = cp.norm(a2) + gamma2 * (cp.norm(eta2,1))
cons2 = [Y@(X@a2-b2) + eta2 >= 1, eta2 >= 0]
```

```python
problem2 = cp.Problem(cp.Minimize(obj2), cons2)
print(problem2.solve(verbose = True, solver = cp.ECOS))
print("\nOptimal a: ", a2.value, "\nOptimal b:", b2.value)
```

```
===============================================================================
                                     CVXPY
                                     v1.4.2
===============================================================================
(CVXPY) Mar 20 09:52:49 AM: Your problem has 181 variables, 2 constraints, and 0
parameters.
(CVXPY) Mar 20 09:52:49 AM: It is compliant with the following grammars: DCP,
DQCP
(CVXPY) Mar 20 09:52:49 AM: (If you need to solve this problem multiple times,
but with different data, consider using parameters.)
(CVXPY) Mar 20 09:52:49 AM: CVXPY will first compile your problem; then, it will
invoke a numerical solver to obtain a solution.
(CVXPY) Mar 20 09:52:49 AM: Your problem is compiled with the CPP
canonicalization backend.
-------------------------------------------------------------------------------
                                  Compilation
-------------------------------------------------------------------------------
(CVXPY) Mar 20 09:52:49 AM: Compiling problem (target solver=ECOS).
(CVXPY) Mar 20 09:52:49 AM: Reduction chain: Dcp2Cone -> CvxAttr2Constr ->
ConeMatrixStuffing -> ECOS
(CVXPY) Mar 20 09:52:49 AM: Applying reduction Dcp2Cone
(CVXPY) Mar 20 09:52:49 AM: Applying reduction CvxAttr2Constr
(CVXPY) Mar 20 09:52:49 AM: Applying reduction ConeMatrixStuffing
(CVXPY) Mar 20 09:52:49 AM: Applying reduction ECOS
(CVXPY) Mar 20 09:52:49 AM: Finished problem compilation (took 9.276e-03
seconds).
-------------------------------------------------------------------------------
                                Numerical solver
-------------------------------------------------------------------------------
(CVXPY) Mar 20 09:52:49 AM: Invoking solver ECOS  to obtain a solution.
-------------------------------------------------------------------------------
                                    Summary
-------------------------------------------------------------------------------
(CVXPY) Mar 20 09:52:49 AM: Problem status: optimal
(CVXPY) Mar 20 09:52:49 AM: Optimal value: 8.944e+01
(CVXPY) Mar 20 09:52:49 AM: Compilation took 9.276e-03 seconds
(CVXPY) Mar 20 09:52:49 AM: Solver (including time spent in interface) took
3.277e-03 seconds
89.43653660717314

Optimal a:  [ 0.20864823 -0.97870147 -1.62007281 -0.4604091   3.76855067]
Optimal b: [-9.24105061]
```

[4]:
```python
gamma3 = 10
a3 = cp.Variable(n, 'a3')
b3 = cp.Variable(1,'b3')
eta3 = cp.Variable(m, 'eta3')
obj3 = cp.norm(a3) + gamma3 * (cp.norm(eta3,1))
cons3 = [Y@(X@a3-b3) + eta3 >= 1, eta3 >= 0]
problem3 = cp.Problem(cp.Minimize(obj3), cons3)
print(problem3.solve(verbose = True, solver = cp.ECOS))
print("\nOptimal a: ", a3.value, "\nOptimal b:", b3.value)
```

```
===============================================================================
                                     CVXPY
                                     v1.4.2
===============================================================================
(CVXPY) Mar 20 09:52:49 AM: Your problem has 181 variables, 2 constraints, and 0
parameters.
(CVXPY) Mar 20 09:52:49 AM: It is compliant with the following grammars: DCP,
DQCP
(CVXPY) Mar 20 09:52:49 AM: (If you need to solve this problem multiple times,
but with different data, consider using parameters.)
(CVXPY) Mar 20 09:52:49 AM: CVXPY will first compile your problem; then, it will
invoke a numerical solver to obtain a solution.
(CVXPY) Mar 20 09:52:49 AM: Your problem is compiled with the CPP
canonicalization backend.
-------------------------------------------------------------------------------
                                  Compilation
-------------------------------------------------------------------------------
(CVXPY) Mar 20 09:52:49 AM: Compiling problem (target solver=ECOS).
(CVXPY) Mar 20 09:52:49 AM: Reduction chain: Dcp2Cone -> CvxAttr2Constr ->
ConeMatrixStuffing -> ECOS
(CVXPY) Mar 20 09:52:49 AM: Applying reduction Dcp2Cone
(CVXPY) Mar 20 09:52:49 AM: Applying reduction CvxAttr2Constr
(CVXPY) Mar 20 09:52:49 AM: Applying reduction ConeMatrixStuffing
(CVXPY) Mar 20 09:52:49 AM: Applying reduction ECOS
(CVXPY) Mar 20 09:52:49 AM: Finished problem compilation (took 1.229e-02
seconds).
-------------------------------------------------------------------------------
                                Numerical solver
-------------------------------------------------------------------------------
(CVXPY) Mar 20 09:52:49 AM: Invoking solver ECOS  to obtain a solution.
-------------------------------------------------------------------------------
                                    Summary
-------------------------------------------------------------------------------
(CVXPY) Mar 20 09:52:49 AM: Problem status: optimal
(CVXPY) Mar 20 09:52:49 AM: Optimal value: 8.524e+02
(CVXPY) Mar 20 09:52:49 AM: Compilation took 1.229e-02 seconds
(CVXPY) Mar 20 09:52:49 AM: Solver (including time spent in interface) took
3.183e-03 seconds
```

```
852.4305773711759

Optimal a:  [ 0.15062914 -0.91314802 -1.52389243 -0.4642144   4.82133807]
Optimal b: [-8.80471718]
```

Predicting

First we load the test data. As above, we make a matrix Y_{test}.

[5]:
```
Xtest = mat['features_test']
ytest = mat['labels_test']
mtest, ntest = Xtest.shape
Ytest = np.zeros((mtest,mtest),float)
for i in range(mtest):
    Ytest[i][i] = ytest[i][0]
```

We only need to find which side of the hyperplane $\{x \mid x^\top a = b\}$ the test data points are - this is obtained by checking whether $y_j = sgn(x_j^\top a - b)$, or equivalently, $y_j \cdot (x_j^\top a - b) > 0$. So again we consider the vector $Y_{\text{test}}(X_{\text{test}} a - b\mathbf{1})$ and find out how many of them have non-positive entries - the lower this number, the better is the prediction.

[6]:
```
print(sum(Ytest@(Xtest@a1.value-b1.value)<=0))
print(sum(Ytest@(Xtest@a2.value-b2.value)<=0))
print(sum(Ytest@(Xtest@a3.value-b3.value)<=0))
```

```
1
2
2
```

[7]:
```
a1.value
```

[7]:
```
array([ 0.14105247,  0.18277618, -0.73224986, -0.10977297,  0.38083898])
```