E_worksheet

February 19, 2024

1 Worksheet E

variationalform https://variationalform.github.io/

Just Enough: progress at pace https://variationalform.github.io/

https://github.com/variationalform

Simon Shaw https://www.brunel.ac.uk/people/simon-shaw.

This document uses python

and also makes use of LaTeX

in Markdown

1.1 What this is about:

This worksheet is based on the material in the notebooks

- probstat
- regress
- svm
- percep
- pca

Note that while the 'lecture' notebooks are prefixed with `1_`, `2_` and so on, to indicate the order in which they should be studied, the worksheets are prefixed with `A_`, `B_`, …

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1.1.1 Exercise 1

A fair dice is thrown: What is the probability that

- it falls on a six?
- it falls on an even number?
- it falls on an odd number greater than 2?

1.1.2 Exercise 2

The defence counsel in the following case has called you as an expert witness.

A woman is on trial for murder. There is no evidence against her except for a lab-reported DNA match between her and a scene-of-crime sample. The prosecution have correctly stated that the theoretical probability of a match between two randomly selected people is one in a million, and thereby conclude that the probability the defendant is innocent is 10^{-6}.

Your legal team have learned that, while the lab never fails to identify a true match, it falsely reports a match once in every $100,000$ *tests (due, for example, to technology limitations, human error, …). The court is also aware that there are around two million other women who fit the criminal profile and, therefore, could have committed the crime.*

What do you advise the defense counsel?

1.1.3 Exercise 3

When studying regression we defined the notion of cost as, for example, total squared error, TSE, with **regularization**:

$$\mathcal{E}(\boldsymbol{\theta}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2 + \alpha\|\boldsymbol{\theta}\|_p^p$$

for $p = 2$ (ridge) or $p = 1$ (LASSO). This regularization term is often referred to as a **penalty term** in the sense that it **penalises** large coeffcients.

Given that we are trying to minimise cost it is clear that with $\alpha > 0$ we will also be trying to minimize $\alpha\|\boldsymbol{\theta}\|_p^p$ which, in turn, tends to introduce small coefficients.

More than that though, the choice $p = 1$ may introduce sparsity into the coefficent vector. This is a form of **feature selection**.

We can see why the ℓ_1 norm seems to promote sparsity by borrowing from the discussion here: https://stats.stackexchange.com/questions/45643/why-l1-norm-for-sparse-models

1. Define the vector $\boldsymbol{v} = (1, \epsilon)^T$ for some small $\epsilon > 0$ and show that $\|\boldsymbol{v}\|_1 = 1+\epsilon$ and $\|\boldsymbol{v}\|_2^2 = 1+\epsilon^2$.
2. Assume that regularization reduces the first coefficent by δ (with $0 < \delta < \epsilon < 1$) so that $\boldsymbol{v}$ becomes $\boldsymbol{v} = (1 - \delta, \epsilon)^T$. Show that now,

$$\|\boldsymbol{v}\|_1 = 1 - \delta + \epsilon \qquad \text{and} \qquad \|\boldsymbol{v}\|_2^2 = 1 - 2\delta + \delta^2 + \epsilon^2.$$

3. Now, assume instead that regularization reduces the second coefficent by δ (with $0 < \delta < \epsilon < 1$) so that $\boldsymbol{v}$ becomes $\boldsymbol{v} = (1, \epsilon - \delta)^T$. Show that now,

$$\|\boldsymbol{v}\|_1 = 1 + \epsilon - \delta \qquad \text{and} \qquad \|\boldsymbol{v}\|_2^2 = 1 + \epsilon^2 - 2\epsilon\delta + \delta^2.$$

4. Is the reduction in norm the same in both cases for the ℓ_1 norm?
5. Is the reduction in norm the same in both cases for the ℓ_2 norm? If not, for which is the reduction largest?
6. Explain what the author on the website (as above) means when s/he says

it's not so much that ℓ_1 penalties encourage sparsity, but that ℓ_2 penalties in some sense discourage sparsity…

1.1.4 Exercise 4

Find the equation of the straight line that passes between the points $\boldsymbol{q} = (q_1, q_2)$ and $\boldsymbol{r} = (r_1, r_2)$ in the (x_1, x_2) plane.

The minimum distance between each point and the line should be the same.

Find the general form of the equation of the line. And give the specific form in the case $\boldsymbol{q} = (5, 1)$ and $\boldsymbol{r} = (2, 3)$.

1.1.5 Exercise 5

Write the straight line equation $x_2 = x_1 + 1$ in the form $\boldsymbol{w} \cdot \boldsymbol{x} = \phi$.

1.1.6 Exercise 6

Find the shortest distance from the point Q at coordinates $(2, 1)$ to the line $\boldsymbol{w} \cdot \boldsymbol{x} = \phi$ with $\boldsymbol{w} = (-1, 1)^T$ and $\phi = 1$.

Caluclate this by hand, and use python.

1.1.7 Exercise 7

1. Use `x1 = np.arange(-2,3,1)` to create a one dimentsional grid in the `x1` direction. Print `x1` to understand what is happening.
2. Use `x2 = np.arange(-2,3,1)` to create a one dimentsional grid in the `x2` direction. Print `x2` to understand what is happening.
3. Use these lines of code to create 2D grids giving `x1` coordinates and `x2` coordinates in each variable:

```
import numpy.matlib
N = x1.shape[0]
X1grid = np.matlib.repmat(x1,N,1)
X2grid = np.matlib.repmat(x2,N,1).T
```

Again, print your results to understand what is happening. Can you see why the `.T` is used for transpose?

1.1.8 Exercise 8

Continuing from the last question, use this code in a new cell to evaluate the function $y = x_1 + x_2$ on the 2D grid. Use the `indx` variable to plot the function in blue when $y < 0$ and in red otherwise. Print out y and `indx` so you can see exactly what is going on.

```
y = X1grid + X2grid
print(y)
indx = y < 0
print(indx)
```

```
plt.scatter(X1grid[indx],X2grid[indx], color='blue')
plt.scatter(X1grid[~indx],X2grid[~indx], color='red')
```

1.1.9 Exercise 9

Replace `plt.scatter` with `plt.plot` in the above (in a new cell). What happens?

Can you solve this by replacing `color='blue'` with `.b` (similarly with `.r`)?

1.1.10 Exercise 10

Flatten the grids and the index with

```
X1f = X1grid.flatten()
X2f = X2grid.flatten()
indxf = indx.flatten()
```

Print out these new quantites - what has happened?

Use these flattened arrays with `plt.scatter` and `plt.plot` as above. Are the plots the same?

1.2 Exercise 11

Design a two-input, one-output perceptron, with a Heaviside activation function, that has decision boundary $4 - 7x_1 + 3x_2 = 0$.

Evaluate the output of the perceptron for the inputs $\boldsymbol{x} = (-4, 3)^T$ and $\boldsymbol{x} = (6, 2)^T$.

Plot this decision boundary with a solid black line over $0.5 \leq x_1 \leq 1.5$.

Also, illustrate this decision boundary for $0 \leq x_1, x_2 \leq 2$ using a background grid of points (as above) with grid spacing 0.1 and two colours.

1.3 Exercise 12

Design a two-input, two-output Heaviside-activated neural network that has decision boundaries $5x_1 + 3x_2 + 6 = 0$ and $4x_1 + 7x_2 + 3 = 0$.

Evaluate the output of the perceptron for the inputs $\boldsymbol{x} = (-4, 3)^T$, $\boldsymbol{x} = (6, 2)^T$, $\boldsymbol{x} = (-6, 2)^T$ and $\boldsymbol{x} = (6, -6)^T$.

Plot these decision boundaries with dotted black lines over $-7 \leq x_1 \leq 4$.

Also, illustrate these decision boundaries for $-10 \leq x_1, x_2 \leq 10$ using a background grid of points (as above) with grid spacing 1 and four colours.

1.4 Exercise 13

A binary classifier gives the following predicted outputs on the test set:

test label

pass

pass

fail

pass

fail

pass

fail

pass

fail

pass

pass

fail

pass

fail

pass

fail

fail

pass

fail

predicted

pass

fail

pass

fail

pass

fail

pass

fail

pass

fail

pass

fail

fail

pass

pass

fail

pass

pass

fail

Create the confusion matrix first by hand, and then using code.

Exercise 14

Suppose you have these data points: $(\boldsymbol{x}_1, \boldsymbol{x}_2) = (3,3),\ (-3,-3)$ corresponding to the features $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

Perform a principal component analysis (PCA). Before actually computing any results, think about what you are expecting to find.

Suppose $(3,3)$ is removed but two more points are added: $(1,2),\ (2,1)$. Repeat the PCA. What do you find now?

In each case, determine the coordinates of these points in the principal component system. Discuss the empirical variance in the data sets and the relevance of the eigenvalues.

2 Outline solutions

As usual - try the above before looking at these…

2.0.1 Answer 1

Here $\Omega = \{1,2,3,4,5,6\}$ and $\mathcal{E}$ has $2^{|\Omega|} = 2^6$ elements.

Appealing to the symmetry of the dice and the impartiality of the laws of physics, we have that

- $\text{Prob}(\{6\}) = 1/6$, because all six numbers are equally likely and exactly one must occur.
- $\text{Prob}(\{2,4,6\}) = 3/6 = 1/2$ or
  - $\text{Prob}(\{2\}) + \text{Prob}(\{4\}) + \text{Prob}(\{6\}) = 1/6 + 1/6 + 1/6 = 1/2$.
- $\text{Prob}(\{3,5\}) = 2/6 = 1/3$.
  - $\text{Prob}(\{3,5\}) = \text{Prob}(\{1,3,5\}) \times \text{Prob}(\{3,4,5,6\}) = 3/6 \times 4/6 = 1/3$.

The second illustrates that

- $\text{P}(A \text{ and } B) = \text{P}(A) + \text{P}(B)$ when A and B are *mutually exclusive*.

2.0.2 Answer 2

The way to discredit the prosecution's probability is to focus on the DNA testing lab's **failure rate**. Let M be the event of a reported match between two samples. Let T be the event of a true match, and let F be the event of a falsely reported match. Then, $P(T) = 10^{-6}$, $P(M) = P(T \cup F) = P(T) + P(F) = 10^{-6} + 10^{-5}$ and so,

$$\mathrm{P}(T \mid M) = \frac{P(T \cap M)}{P(M)} = \frac{P(T)}{P(M)} = \frac{10^{-6}}{10^{-6} + 10^{-5}} = \frac{1}{11}.$$

Another way of saying this is that, if we test every one of the $2,000,000$ women who may be guilty, we can expect 2 (one in a million) to show a true match and the lab to falsely report another 20 (one in $100,000$) as also matching. Hence, of the 22 who are reported to match, only 2 actually match and $2/22 = 1/11 < 10\%$ as above.

This means, we can imagine telling the jury, that there is a less than 10% probability that the defendant's DNA matches the DNA found at the crime scene. Moreover, in the absence of other evidence, there are 22 other defendant candidates and, by choosing one at random, the probability that we have the guilty one is only 1/22, or about 4.5%.

2.0.3 Answer 3

1. $\boldsymbol{v} = (1, \epsilon)^T$ and so $\|\boldsymbol{v}\|_1 = \sum_i |v_i| = 1 + \epsilon$ and $\|\boldsymbol{v}\|_2^2 = \sum_i |v_i|^2 = 1 + \epsilon^2$.
2. $\boldsymbol{v} = (1 - \delta, \epsilon)^T$ and so $\|\boldsymbol{v}\|_1 = \sum_i |v_i| = 1 - \delta + \epsilon$ and $\|\boldsymbol{v}\|_2^2 = \sum_i |v_i|^2 = (1 - \delta)^2 + \epsilon^2 = 1 - 2\delta + \delta^2 + \epsilon^2$.
3. $\boldsymbol{v} = (1, \epsilon - \delta)^T$ and so $\|\boldsymbol{v}\|_1 = \sum_i |v_i| = 1 + \epsilon - \delta$ and $\|\boldsymbol{v}\|_2^2 = \sum_i |v_i|^2 = 1 + (\epsilon - \delta)^2 = 1 + \epsilon^2 - 2\epsilon\delta + \delta^2$.
4. Yes, the norm is reduced by δ in both cases.
5. The ℓ_2 norm is reduced by $(2-\delta)\delta$ when the larger component is reduced, but only by $(2\epsilon-\delta)\delta$ when the smaller component is reduced.
6. Sparsity results from zeros being introduced into the coeffcient vector. A small value need only be reduced by a correspondingly small amount to become zero. The argument above suggests that small values only get a correspondingly small reduction in the ℓ_2 norm, but get equal reduction as larger values in the ℓ_1 norm. This leads us to believe that there is an increased possibility of zeroes being introduced in the LASSO case.

2.0.4 Answer 4

The midpoint is $\hat{\boldsymbol{x}} = (\hat{x}_1, \hat{x}_2) = \frac{1}{2}\Big(\boldsymbol{q} + \boldsymbol{r}\Big)$ and the straight line must go through that point.

The straight line must be orthogonal to $\boldsymbol{q} - \boldsymbol{r}$ and so its gradient must be $m = (q_1 - r_1)/(r_2 - q_2)$.

This line has the general form of equation,

$$x_2 - \hat{x}_2 = \left(\frac{q_1 - r_1}{r_2 - q_2}\right)(x_1 - \hat{x}_1).$$

In the specific case given we have $\hat{\boldsymbol{x}} = (\hat{x}_1, \hat{x}_2) = \frac{1}{2}\Big(5 + 2, 1 + 3\Big) = \frac{1}{2}(7, 4)$, and with $\boldsymbol{q} = (5, 1)$ and $\boldsymbol{r} = (2, 3)$ we have,

$$x_2 - \frac{4}{2} = \left(\frac{5 - 2}{3 - 1}\right)\left(x_1 - \frac{7}{2}\right).$$

Or, simplified,

$$x_2 = \frac{3}{2}x_1 - \frac{3}{2}\frac{7}{2} + \frac{4}{2} = \frac{3}{2}x_1 - \frac{21}{4} + \frac{8}{4} = \frac{3}{2}x_1 - \frac{13}{4}.$$

2.0.5 Answer 5

We have $(w_1, w_2)^T \cdot (x_1, x_2)^T = \phi$ with $(w_1, w_2)^T = (-1, 1)^T$ and $\phi = 1$.

2.0.6 Answer 6

We use the formula we derived. With $\boldsymbol{q} = (2, 1)^T$,

$$d = \frac{\phi - \boldsymbol{w}^T\boldsymbol{q}}{\|\boldsymbol{w}\|_2} = \frac{1 - (-1, 1)(2, 1)^T}{\sqrt{2}} = \frac{2}{\sqrt{2}}$$

and so $d = \sqrt{2}$.

[2]:
```
# Answer 6
w = np.array([[-1],[1]])
q = np.array([[2],[1]])
phi = 1
print(w)
print(q)
print(np.linalg.norm(w))
d = (phi - np.ndarray.item(w.T.dot(q)))/np.linalg.norm(w)
print(d)
d = (phi - np.matmul(w.T,q))/np.linalg.norm(w)
print(d)
d = (phi - np.vdot(w.T,q))/np.linalg.norm(w)
print(d)
```

```
[[-1]
 [ 1]]
[[2]
 [1]]
1.4142135623730951
1.414213562373095
[[1.41421356]]
1.414213562373095
```

2.0.7 Answer 7

Here is a fully worked example

[3]:
```
x1 = np.arange(-2,3,1)
x2 = np.arange(-2,3,1)
print(x1, x2)
import numpy.matlib
```

```
N = x1.shape[0]
X1grid = np.matlib.repmat(x1,N,1)
X2grid = np.matlib.repmat(x2,N,1).T
print('X1grid = \n', X1grid, '\n and X2grid = \n', X2grid)
```

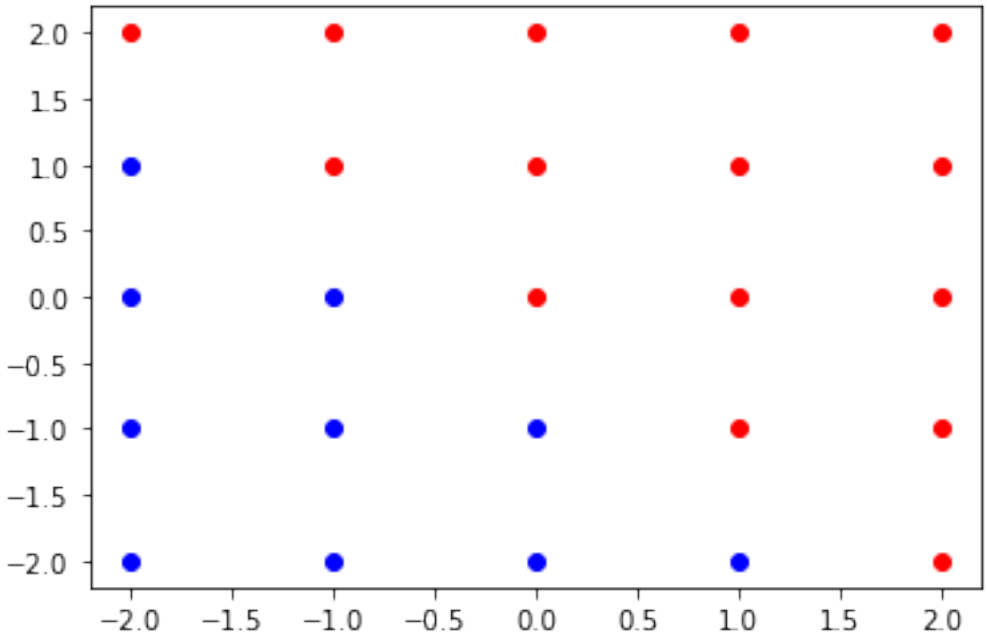
```
[-2 -1  0  1  2] [-2 -1  0  1  2]
X1grid =
 [[-2 -1  0  1  2]
 [-2 -1  0  1  2]
 [-2 -1  0  1  2]
 [-2 -1  0  1  2]
 [-2 -1  0  1  2]]
 and X2grid =
 [[-2 -2 -2 -2 -2]
 [-1 -1 -1 -1 -1]
 [ 0  0  0  0  0]
 [ 1  1  1  1  1]
 [ 2  2  2  2  2]]
```

2.0.8 Answer 8

Here is a fully worked example

[4]:
```
y = X1grid + X2grid
indx = y < 0
plt.scatter(X1grid[indx],X2grid[indx], color='blue')
plt.scatter(X1grid[~indx],X2grid[~indx], color='red')
#print(y)
#print(indx)
```

[4]:
```
<matplotlib.collections.PathCollection at 0x7fbcc83b29b0>
```

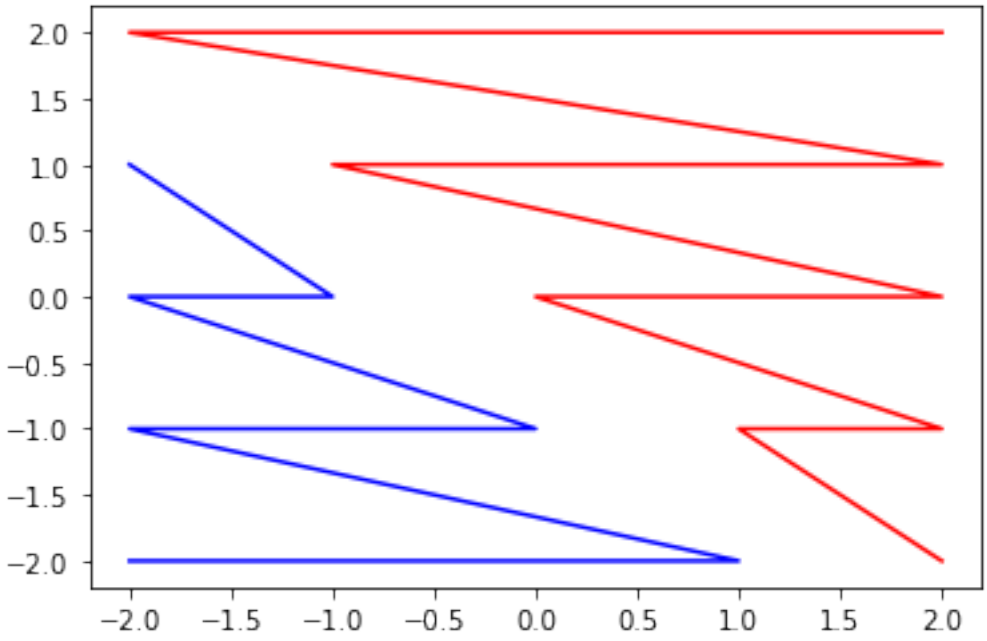


2.0.9 Answer 9

Here is a fully worked example

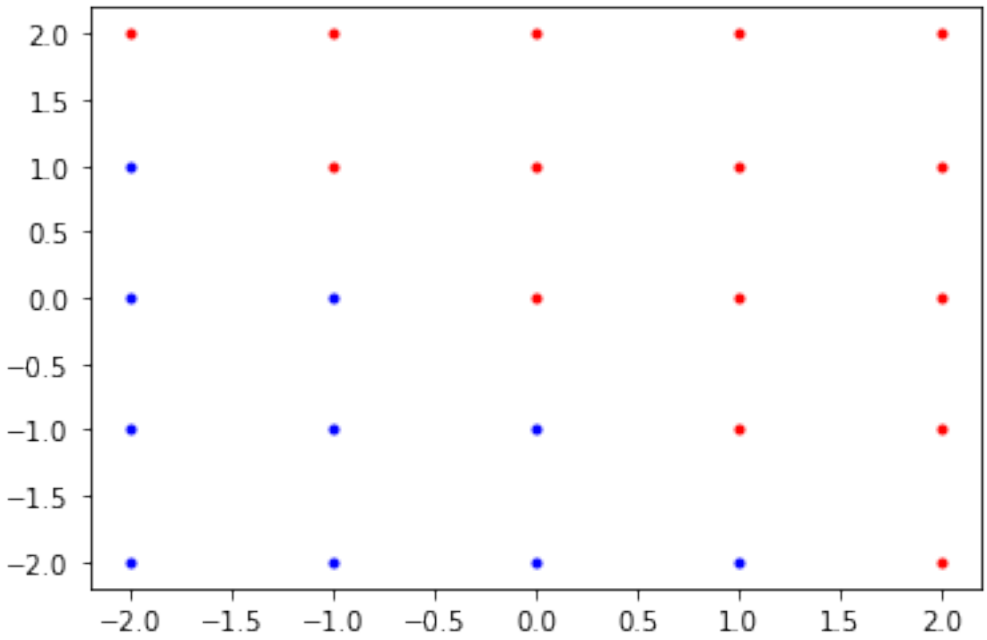
[5]:
```
plt.plot(X1grid[indx],X2grid[indx], color='blue')
plt.plot(X1grid[~indx],X2grid[~indx], color='red')
```

[5]:
```
[<matplotlib.lines.Line2D at 0x7fbcb9649550>]
```



[6]:
```
plt.plot(X1grid[indx],X2grid[indx], '.b')
plt.plot(X1grid[~indx],X2grid[~indx], '.r')
```

[6]: [<matplotlib.lines.Line2D at 0x7fbc989df978>]



2.0.10 Answer 10

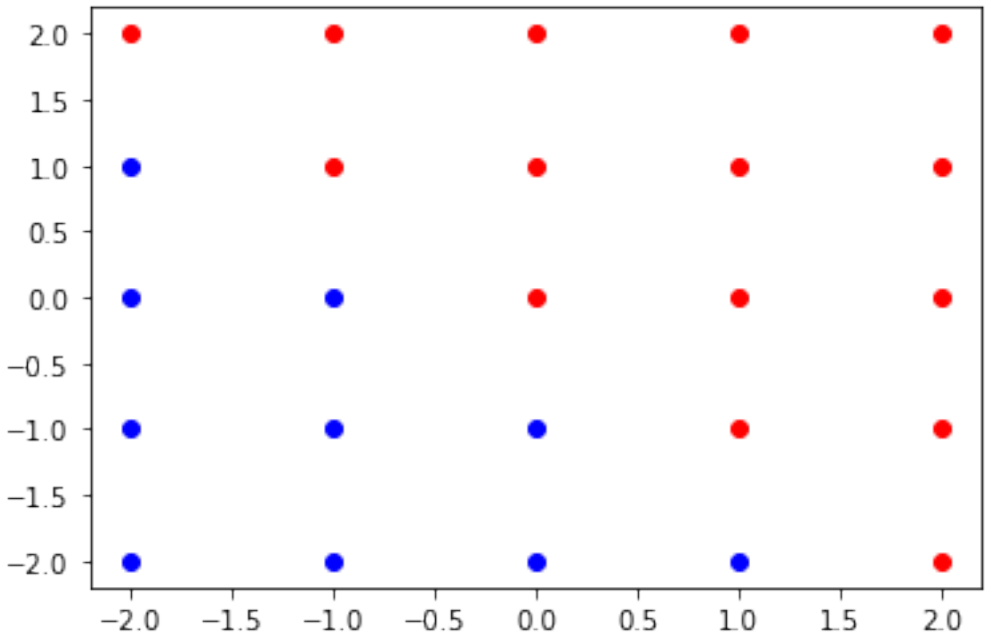
Here is a fully worked example.

[7]:
```
X1f = X1grid.flatten()
X2f = X2grid.flatten()
indxf = indx.flatten()
print('X1f = \n', X1f)
print('X2f = \n', X2f)
print('indxf = \n', indxf)
```

```
X1f =
 [-2 -1  0  1  2 -2 -1  0  1  2 -2 -1  0  1  2 -2 -1  0  1  2 -2 -1  0  1
  2]
X2f =
 [-2 -2 -2 -2 -2 -1 -1 -1 -1 -1  0  0  0  0  0  1  1  1  1  1  2  2  2  2
  2]
indxf =
 [ True  True  True  True False  True  True  True False False  True  True
 False False False  True False False False False False False False False
 False]
```

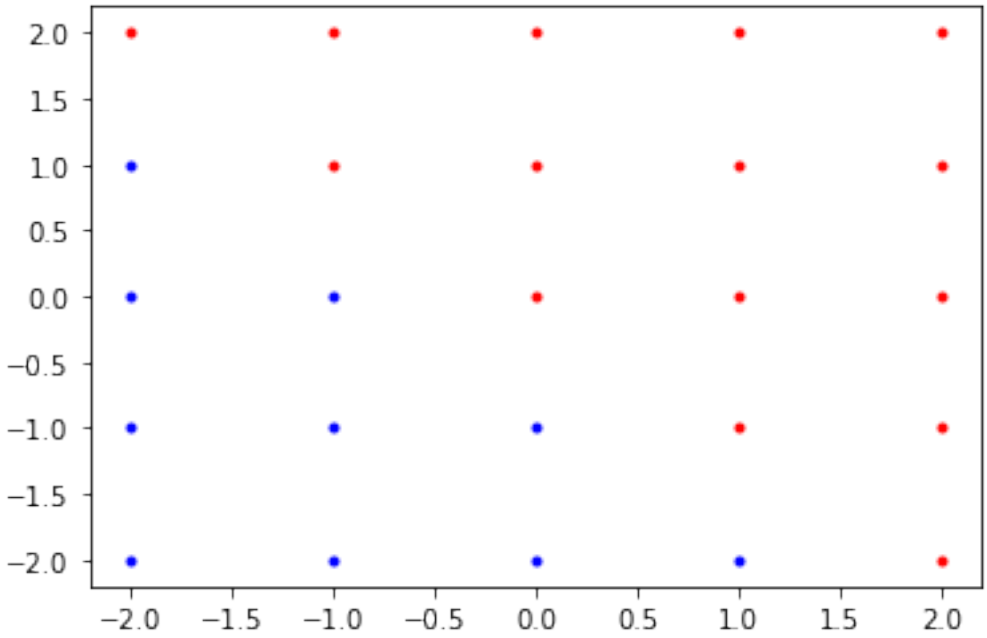
[8]:
```
plt.scatter(X1f[indxf],X2f[indxf], color='blue')
plt.scatter(X1f[~indxf],X2f[~indxf], color='red')
```

[8]:
```
<matplotlib.collections.PathCollection at 0x7fbc98a1a898>
```



[9]:
```
plt.plot(X1f[indxf],X2f[indxf], '.b')
plt.plot(X1f[~indxf],X2f[~indxf], '.r')
```

[9]:
```
[<matplotlib.lines.Line2D at 0x7fbc98a9e080>]
```



2.0.11 Answer 11

We have in general...

$$\boldsymbol{x} = (x_0, x_1, \ldots, x_N)^T, \boldsymbol{y} = (y_0, y_1, \ldots, y_M)^T, \boldsymbol{y} = \sigma(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$$

Here for a decision boundary of $-7x_1 + 3x_2 + 4 = 0$ we choose $\boldsymbol{W} = (-7, 3)^T$ and $b = 4$. Then

$$y = \mathcal{H}(\boldsymbol{W}^T\boldsymbol{x} + b).$$

Note that both y and b are scalars here. The equation of the decision in boundary can also be written as $x_2 = (7x_1 - 4)/3$.

[10]:
```
W = np.array([[-7,3]]).T
b = 4
# find y for input x = (-4,3)
X = np.array([[-4,3]]).T
y = np.heaviside(W.T@X+b,0)
print('For input x = (-4,3), y = ', y)
# find y for input x = (6,2)
```

```
X = np.array([[6,2]]).T
y = np.heaviside(W.T@X+b,0)
print('For input x = ( 6,2), y = ', y)
```

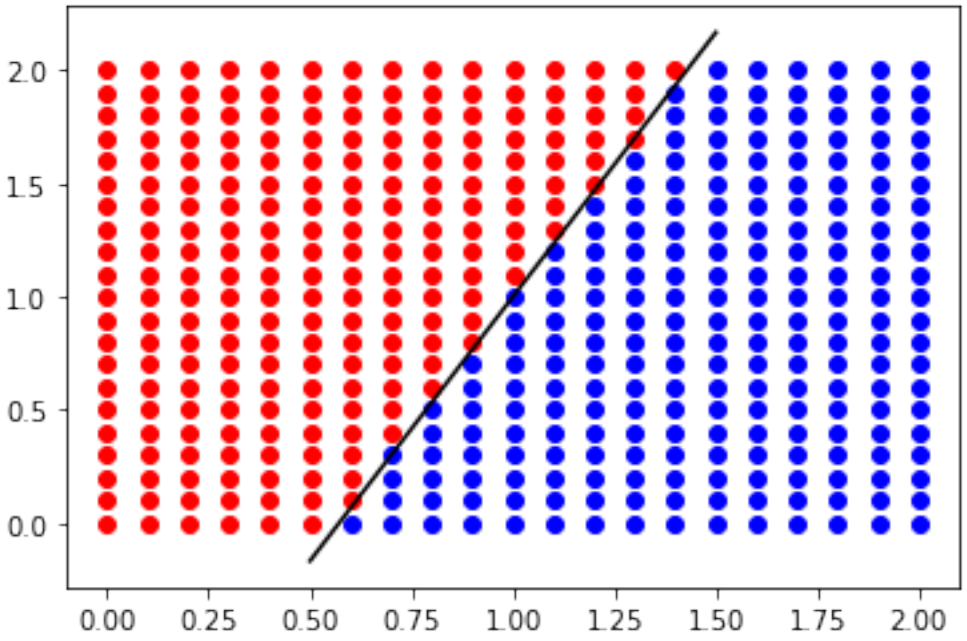
```
For input x = (-4,3), y =  [[1.]]
For input x = ( 6,2), y =  [[0.]]
```

[11]:
```
s = 0.1
x1 = np.arange(0,2+s,s)
x2 = np.arange(0,2+s,s)
X = np.zeros([2,1])
N = x1.shape[0]
y = np.zeros([N,N])
import numpy.matlib
X1grid = np.matlib.repmat(x1,N,1)
X2grid = np.matlib.repmat(x2,N,1).T

for i in range(N):
  for j in range(N):
    X[0] = X1grid[i,j]
    X[1] = X2grid[i,j]
    y[i,j] = np.heaviside(W.T @ X + b, 0)
```

[12]:
```
indx = (y < 0.5)
plt.scatter(X1grid[ indx],X2grid[ indx], color='blue')
plt.scatter(X1grid[~indx],X2grid[~indx], color='red')
P1x=0.5
P2x=1.5
plt.plot([P1x, P2x], [(7*P1x-4)/3, (7*P2x-4)/3], color='black')
```

[12]:
```
[<matplotlib.lines.Line2D at 0x7fbca8b274a8>]
```



2.0.12 Answer 12

We have in general...

$$\boldsymbol{x} = (x_0, x_1, \ldots, x_N)^T, \boldsymbol{y} = (y_0, y_1, \ldots, y_M)^T, \boldsymbol{y} = \sigma(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$$

Here for decision boundaries of $5x_1 + 3x_2 + 6 = 0$ and $4x_1 + 7x_2 + 3 = 0$ we choose $\boldsymbol{W} = \binom{5}{3}\ \binom{4}{7}$ and $\boldsymbol{b} = \binom{6}{3}$. Then

$$\boldsymbol{y} = \mathcal{H}(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}).$$

The equations of the decision in boundaries can also be written as $x_2 = -(5x_1 + 6)/3$ and $x_2 = -(4x_1 + 3)/7$.

[13]:
```
# here is the neural network definition
W = np.array([[5,4],[3,7]])
b = np.array([[6,3]]).T
# find y for input x = (-4,3)
X = np.array([[-4,3]]).T
y = np.heaviside(W.T@X+b,0)
print('For input x = (-4,3), y = \n', y)
# find y for input x = (6,2)
X = np.array([[6,2]]).T
y = np.heaviside(W.T@X+b,0)
print('For input x = ( 6,2), y = \n', y)
```

```
# find y for input x = (-6,2)
X = np.array([[-6,2]]).T
y = np.heaviside(W.T@X+b,0)
print('For input x = (-6,2), y = \n', y)
# find y for input x = (6,-6)
X = np.array([[6,-6]]).T
y = np.heaviside(W.T@X+b,0)
print('For input x = ( 6,-6), y = \n', y)
```

```
For input x = (-4,3), y =
 [[0.]
 [1.]]
For input x = ( 6,2), y =
 [[1.]
 [1.]]
For input x = (-6,2), y =
 [[0.]
 [0.]]
For input x = ( 6,-6), y =
 [[1.]
 [0.]]
```

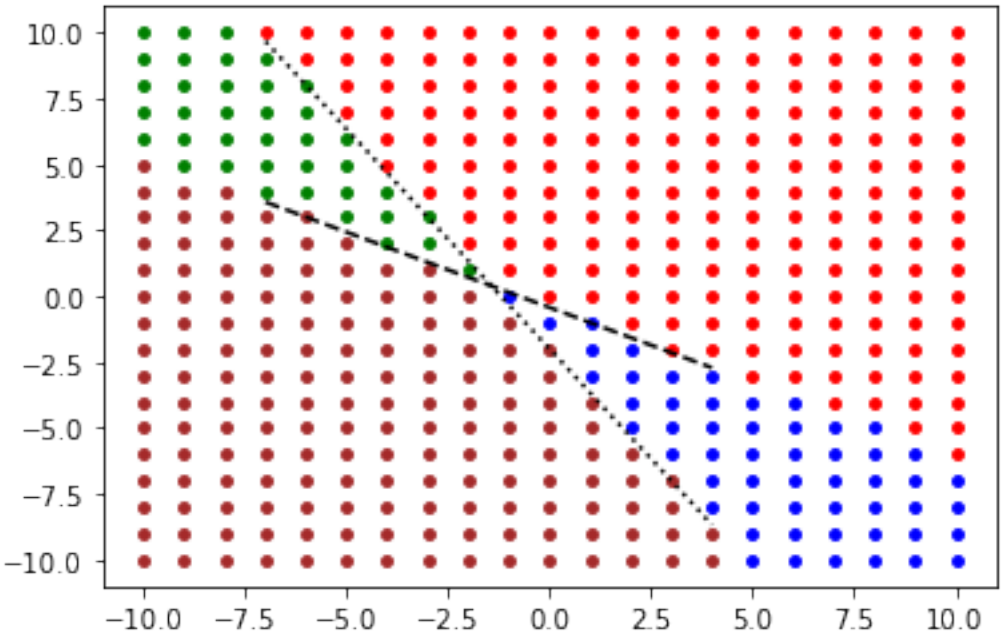
[14]:
```
# two variable, output and input
s = 1
x1 = np.arange(-10,10+s,s)
x2 = np.arange(-10,10+s,s)
X = np.zeros([2,1])
N = x1.shape[0]
y = np.zeros([2,N,N])
import numpy.matlib
X1grid = np.matlib.repmat(x1,N,1)
X2grid = np.matlib.repmat(x2,N,1).T

for i in range(N):
  for j in range(N):
    X[0] = X1grid[i,j]
    X[1] = X2grid[i,j]
    y[:,[i],[j]] = np.heaviside(W.T @ X + b, 0)

indx0 = (y[0,:] < 0.5)
indx1 = (y[1,:] < 0.5)
indx00 = np.logical_and( indx0, indx1)
indx11 = np.logical_and(~indx0,~indx1)
indx01 = np.logical_and( indx0,~indx1)
indx10 = np.logical_and(~indx0, indx1)
```

[15]:
```
plt.scatter(X1grid[indx00], X2grid[indx00],15, color='brown')
plt.scatter(X1grid[indx11], X2grid[indx11],15, color='red')
plt.scatter(X1grid[indx01], X2grid[indx01],15, color='green')
plt.scatter(X1grid[indx10], X2grid[indx10],15 , color='blue')
P1x=-7
P2x=4
plt.plot([P1x, P2x], [-(5*P1x + 6)/3, -(5*P2x + 6)/3], ':k')
plt.plot([P1x, P2x], [-(4*P1x + 3)/7, -(4*P2x + 3)/7], '--k')
```

[15]:
```
[<matplotlib.lines.Line2D at 0x7fbc88324f60>]
```



Answer 13

First we just count them up...

test pass

4

6

test fail

6

3

pred pass

pred fail

In code we can just write,

[16]:
```
y_test = np.array([1,1,0,1,0,1,0,1,0,1,1,0,1,0,1,0,0,1,0])
y_pred = np.array([1,0,1,0,1,0,1,0,1,0,1,0,0,1,1,0,1,1,0])

from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
accsc = accuracy_score(y_test, y_pred)
print("Accuracy:", accsc)
```

```
Confusion Matrix:
[[3 6]
 [6 4]]
Accuracy: 0.3684210526315789
```

Does this surprise you? Why is it not the same?

It's because the values are used to index the matrix. We can alter them like this

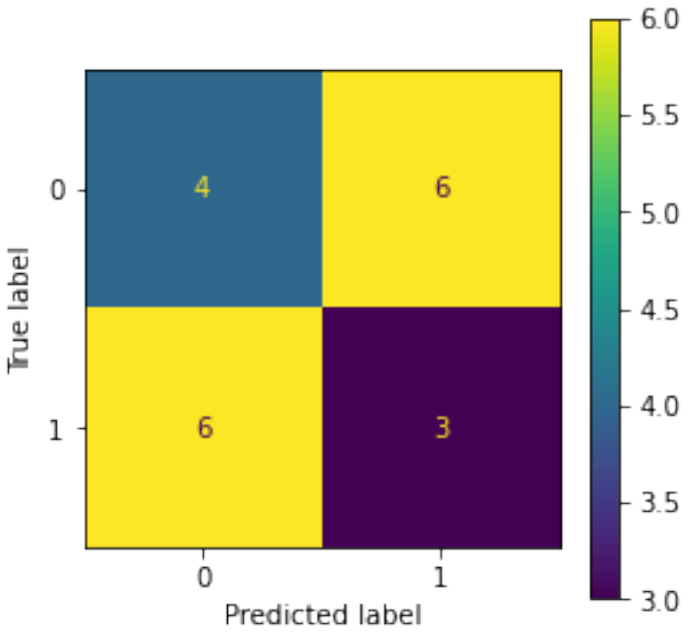
[17]:
```
y_test = ~y_test
y_pred = ~y_pred

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[4 6]
 [6 3]]
```

[18]:
```
from sklearn.metrics import ConfusionMatrixDisplay
cmplot = ConfusionMatrixDisplay(cm, display_labels=range(2))
fig, ax = plt.subplots(figsize=(4,4))
cmplot.plot(ax=ax)
```

[18]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fbca8a9c1d0>
```



It is also possible to use `True` and `False`, or multi=classes, like this:

[19]:
```
y_test = np.array([True, True, False, True, False, False, True])
y_pred = np.array([True, False, True, False, True, False, True])
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
y_test = np.array([1,2,0,1,0,1,0,2,0,1,2,0,1,0,1,2,2,1,0])
y_pred = np.array([1,0,1,0,1,2,1,0,2,0,2,0,2,1,1,0,1,1,0])
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[1 2]
 [2 2]]
Confusion Matrix:
[[2 4 1]
 [2 3 2]
 [3 1 1]]
```

Answer 14

We need the covariance matrix and its eigen-system in each case.

[20]:
```
# Case 1 - data is already centred - check
X = np.array([[3, 3], [-3, -3]])
```

```
print('X = \n',X)
print(f'Column means: col 1, {X[:,0].mean()} and col 2, {X[:,0].mean()}')
# and the empirical covariance matrix
N = 2
S = 1/N*X.T @ X
print('S = \n',S)
# solve the eigenvalue problem||
lmda, V = np.linalg.eig(S)
print('evals = ', lmda)
print('evecs = \n', V)
print('( 3, 3) distance along PC1 = ', X[0,:].T @ V[:,0])
print('(-3,-3) distance along PC1 = ', X[1,:].T @ V[:,0])
print('( 3, 3) distance along PC2 = ', X[0,:].T @ V[:,1])
print('(-3,-3) distance along PC2 = ', X[1,:].T @ V[:,1])
```

```
X =
 [[ 3  3]
 [-3 -3]]
Column means: col 1, 0.0 and col 2, 0.0
S =
 [[9. 9.]
 [9. 9.]]
evals =  [18.  0.]
evecs =
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
( 3, 3) distance along PC1 =  4.242640687119285
(-3,-3) distance along PC1 =  -4.242640687119285
( 3, 3) distance along PC2 =  0.0
(-3,-3) distance along PC2 =  0.0
```

[21]:
```
# Case 2 - data is already centred - check
X = np.array([[-3, -3], [1,2], [2,1]])
print('X = \n',X)
print(f'Column means: col 1, {X[:,0].mean()} and col 2, {X[:,1].mean()}')
# and the empirical covariance matrix
N = 3
S = 1/N*X.T @ X
print('S = \n',S)
# solve the eigenvalue problem
lmda, V = np.linalg.eig(S)
print('evals = ', lmda)
print('evecs = \n', V)
print('(-3,-3) distance along PC1 = ', X[0,:].T @ V[:,0])
print('(-3,-3) distance along PC2 = ', X[0,:].T @ V[:,1])
print('( 1, 2) distance along PC1 = ', X[1,:].T @ V[:,0])
print('( 1, 2) distance along PC2 = ', X[1,:].T @ V[:,1])
```

```
print('( 2, 1) distance along PC1 = ', X[2,:].T @ V[:,0])
print('( 2, 1) distance along PC2 = ', X[2,:].T @ V[:,1])
```

```
X =
 [[-3 -3]
 [ 1  2]
 [ 2  1]]
Column means: col 1, 0.0 and col 2, 0.0
S =
 [[4.66666667 4.33333333]
 [4.33333333 4.66666667]]
evals =  [9.         0.33333333]
evecs =
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
(-3,-3) distance along PC1 =  -4.242640687119286
(-3,-3) distance along PC2 =  -4.440892098500626e-16
( 1, 2) distance along PC1 =  2.1213203435596424
( 1, 2) distance along PC2 =  0.7071067811865477
( 2, 1) distance along PC1 =  2.121320343559643
( 2, 1) distance along PC2 =  -0.7071067811865474
```

In the first case there is just one principal component. The data are one-dimensional. The empirical variance is $2 \times \frac{1}{2}(3^2 + (-3)^2) = 18$ and is all captured by the only non-zero eigenvalue.

In the second case the variance is $2 \times \frac{1}{3}(3^2 + 1^2 + 2^2) = 28/3 \approx 9.333....$

2.1 Technical Notes, Production and Archiving

Ignore the material below. What follows is not relevant to the material being taught.

Production Workflow

- Finalise the notebook material above
- Clear and fresh run of entire notebook
- Create html slide show:
  - `jupyter nbconvert --to slides E_worksheet.ipynb`
- Set `OUTPUTTING=1` below
- Comment out the display of web-sourced diagrams
- Clear and fresh run of entire notebook
- Comment back in the display of web-sourced diagrams
- Clear all cell output
- Set `OUTPUTTING=0` below
- Save
- git add, commit and push to FML
- copy PDF, HTML etc to web site
  - git add, commit and push
- rebuild binder

Some of this originated from

https://stackoverflow.com/questions/38540326/save-html-of-a-jupyter-notebook-from-within-the-n

These lines create a back up of the notebook. They can be ignored.

At some point this is better as a bash script outside of the notebook

[22]:
```
%%bash
NBROOTNAME='E_worksheet'
OUTPUTTING=1

if [ $OUTPUTTING -eq 1 ]; then
  jupyter nbconvert --to html $NBROOTNAME.ipynb
  cp $NBROOTNAME.html ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.html
  mv -f $NBROOTNAME.html ./formats/html/

  jupyter nbconvert --to pdf $NBROOTNAME.ipynb
  cp $NBROOTNAME.pdf ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.pdf
  mv -f $NBROOTNAME.pdf ./formats/pdf/

  jupyter nbconvert --to script $NBROOTNAME.ipynb
  cp $NBROOTNAME.py ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.py
  mv -f $NBROOTNAME.py ./formats/py/
else
  echo 'Not Generating html, pdf and py output versions'
fi
```

```
Not Generating html, pdf and py output versions
```

[ ]:
```

```