Trabajo Práctico Nº1

Métodos de Aprendizaje NO Supervisado

Ribas, Ignacio
Marchetti, Gianfranco

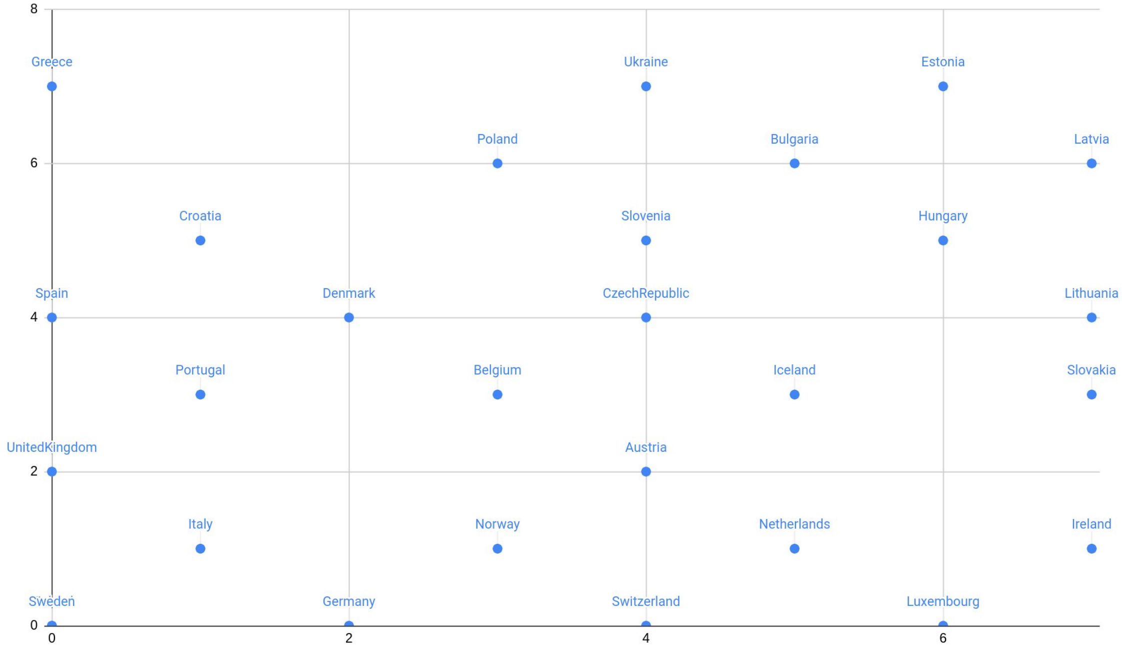
Red de Kohonen

```
class KohonenNetwork:

   def __init__(self, vectorDimension, M, D, randomWeights=False):
       self.vDim = vectorDimension
       self.M = M
       if randomWeights:
           self.W = np.asmatrix(np.random.random(size=(M, vectorDimension)))
       else:
           self.W = np.array([])
           idx = np.random.randint(D.shape[0], size = M)
           self.W = D[idx, :]

       self.L = math.sqrt(M) # The side of the square
```

```python
def train(self, D, lda, R=3):
       indices = list(range(D.shape[0]))
       it = 0
       activationCounts = {i : 0 for i in range(self.M)}
       while it < lda:
           random.shuffle(indices)
           for index in indices:
               curr = D[index]
               distances = [ [np.linalg.norm(curr - x), i] for i, x in enumerate(self.W) ]
               minIndex = min(distances, key=lambda x: x[0])[1]
               activationCounts[minIndex]+=1
               for rowNo in range(self.W.shape[0]):
                   if theta(minIndex, rowNo, it, self.L, R) > 0:
                       self.W[rowNo] = self.W[rowNo] + theta(minIndex, rowNo, it, self.L, R) * alpha(it, lda) *
(curr - self.W[rowNo])
               if it % (lda*.1) == 0 and R > 1:
                   R -= 1
               it+=1
       return activationCounts
   def getClass(self, vector):
       distances = [ [np.linalg.norm(vector - x), i] for i, x in enumerate(self.W) ]
       neuronNo = min(distances, key=lambda x: x[0])[1]
       return Coord(x=neuronNo // self.L, y=neuronNo % self.L)
```

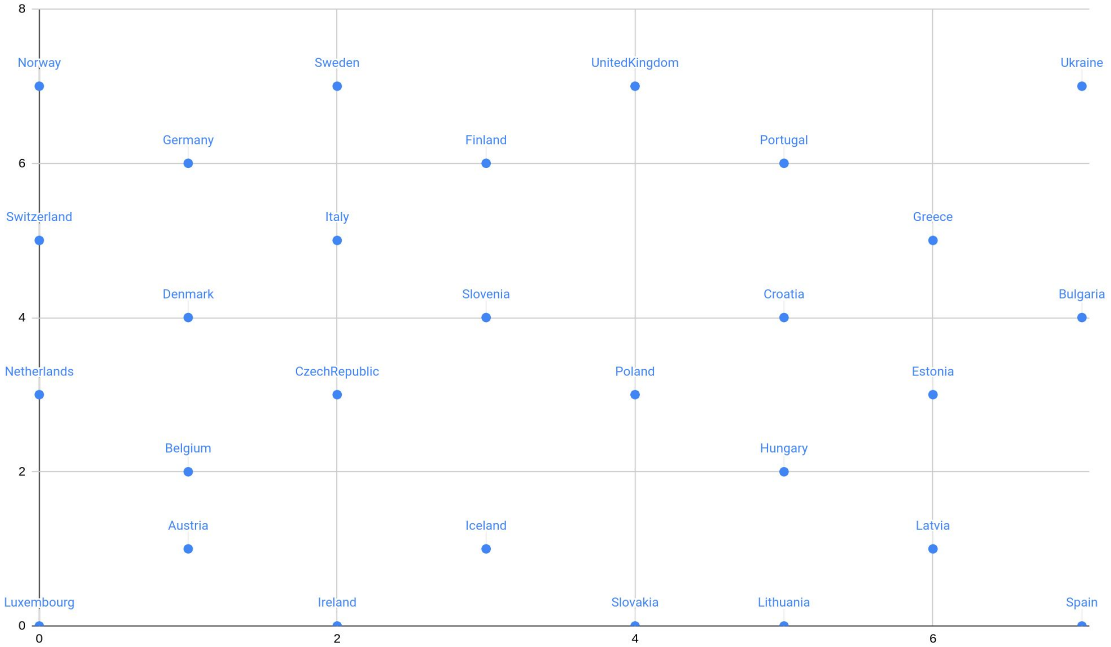
Red de Kohonen

Ejecución 1



Red de Kohonen

Ejecución 2



Red de Kohonen

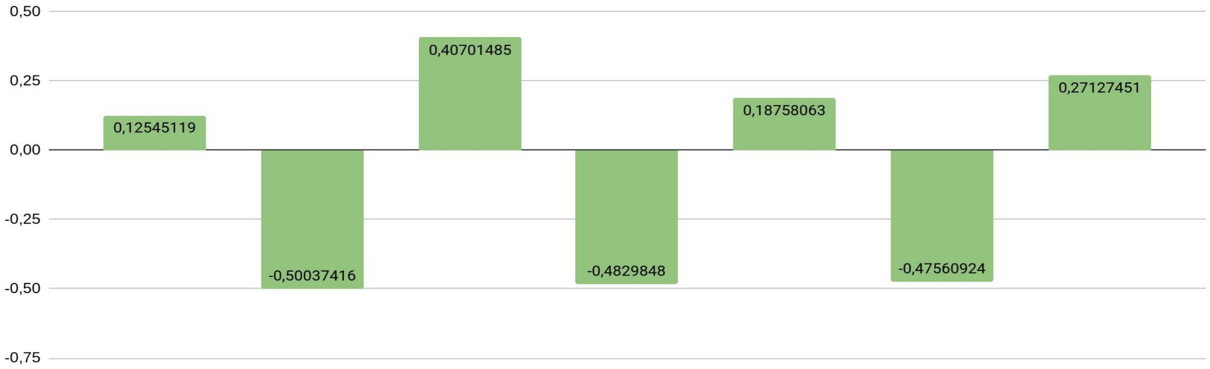
Count Plot 6x6

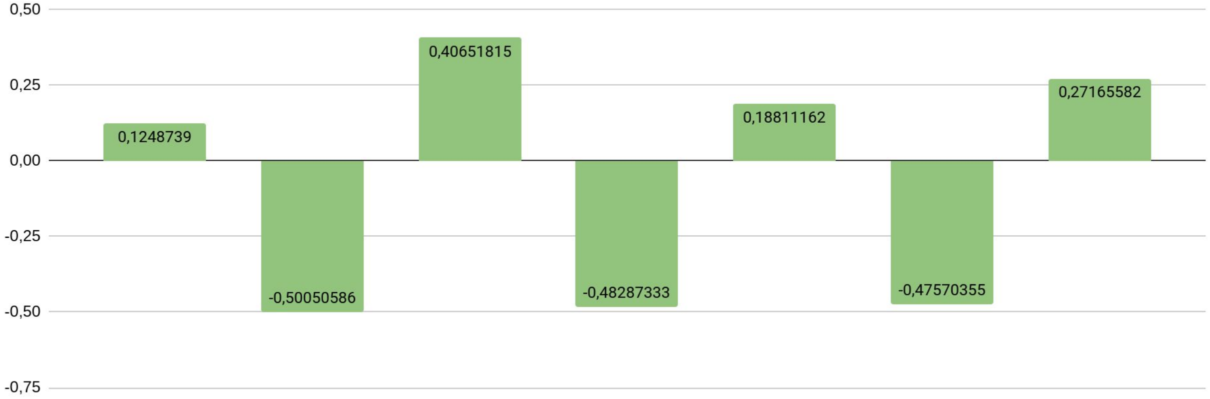
6	200	66	86	94	71	546
5	78	102	66	37	160	166
4	128	47	96	101	104	124
3	196	34	77	104	87	118
2	110	153	164	42	104	97
1	511	312	179	123	93	236
	1	**2**	**3**	**4**	**5**	**6**

Regla de Oja

```
import numpy as np

class OjasRuleNeuron():

   def __init__(self, vectorLength, r=0.01):
       self.w = np.asmatrix(np.random.random(size=(1, vectorLength)))
       self.w /= np.linalg.norm(self.w)
       self.r = r

   def train(self, X, iterations):
       r = self.r
       for it in range(iterations):
           wOld = self.w
           for elemNo in range(X.shape[0]):
               y = np.dot(self.w, X[elemNo].T)
               self.w = self.w + r * y * (X[elemNo] - y * self.w)
           if np.abs(wOld - self.w).all() <= r*100:
               r /= 100
```

Regla de Oja


0,50
0,25
0,00
-0,25
-0,50
-0,75
0,12545119
-0,50037416
0,40701485
-0,4829848
0,18758063
-0,47560924
0,27127451


0,50
0,25
0,00
-0,25
-0,50
-0,75
0,1248739
-0,50050586
0,40651815
-0,48287333
0,18811162
-0,47570355
0,27165582

Regla de Oja

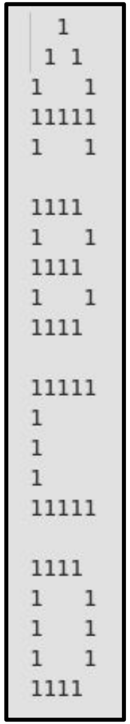
Country	Area	GDP	Pop.growth	Unemployment	Poverty
Austria	-507.835	0.683900	-176.789	-1.245.527	-1.081.748
Belgium	-835.987	0.417061	-115.927	-592.442	-681.094
Hungary	-451.504	-0.860957	-602.821	213.030	1.396.898
Iceland	-390.159	0.438128	1.121.595	-548.903	-1.583.720
Ireland	-591.487	0.627724	2.014.235	974.963	-1.808.918
Italy	829.977	-0.095550	533.265	-331.208	-853.224
Latvia	-626.453	-1.057575	-1.454.886	626.651	2.306.059
Lithuania	-622.079	-0.896068	-805.694	1.192.659	1.530.100
Luxembourg	-1.007.879	3.422512	2.075.096	-918.985	-3.478.435
Netherlands	-768.226	0.711989	675.276	-1.201.988	-1.840.053
Norway	968.157	1.512506	431.829	-1.441.453	-2.106.511
Poland	899.768	-0.818825	-399.949	539.573	1.471.774
Bulgaria	-341.689	-1.268238	-1.860.631	-69.973	2.609.879
Portugal	-457.274	-0.594118	127.520	604.882	526.493
Slovakia	-722.137	-0.601140	-34.778	713.729	782.966
Slovenia	-899.073	-0.214926	-623.109	408.956	67.543
Spain	2.085.115	-0.095550	1.081.021	2.564.138	-163.767
Sweden	1.746.308	0.620702	107.233	-527.133	-885.105
Switzerland	-769.862	0.887541	1.628.777	-1.550.301	-3.281.586
Ukraine	2.689.093	-1.731695	-1.515.748	-440.055	4.580.268
United Kingdom	474.837	0.325774	878.148	-396.516	-340.819
Croatia	-675.636	-0.973310	-420.236	1.693.357	1.270.149
Czech Republic	-538.618	-0.334301	-501.385	-309.438	-167.209
Denmark	-758.685	0.360885	249.243	-831.907	-955.191
Estonia	-745.557	-0.804781	-1.556.322	561.343	2.487.735
Finland	1.056.391	0.290664	-95.640	-461.825	-210.563
Germany	1.172.517	0.438128	-643.396	-853.676	-592.394
Greece	-212.023	-0.390478	-115.927	1.628.049	1.000.472

Hopfield

```python
import numpy as np
class HopfieldNetwork:

    def store(self, examples):
        X = np.asmatrix(examples)
        auxW = [[ (1 / X.shape[1]) * sum([X.item(k, i)*X.item(k, j) for k in range(X.shape[0])]) if i != j else 0 for j in range(X.shape[1]) ] for i in range(X.shape[1])]
        aaux = np.asmatrix(auxW).reshape(X.shape[1], X.shape[1])
        self.w = (1 / X.shape[1]) * np.matmul(X.T, X) - np.eye(X.shape[1])
        self.w = aaux

    def recognize(self, x, iterations=1000):
        S = np.asmatrix(x).T
        oldS = np.asmatrix(np.zeros(S.shape[1])).T
        history = []
        it = 0
        while (oldS - S).any() != 0 and it < iterations:
            oldS = S
            history.append(oldS)
            S = np.sign(np.matmul(self.w, S))
            it+=1
        return S, history
```

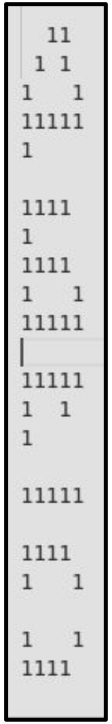
Testing

Conjunto A - B - C - D



```
  1
 1 1
1   1
11111
1   1

1111
1   1
1111
1   1
1111

11111
1
1
1
11111

1111
1   1
1   1
1   1
1111
```

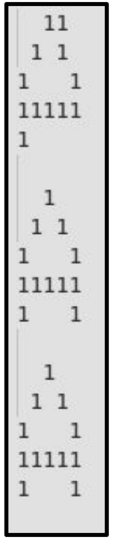
Un poco de ruido



```
  11
 1 1
1   1
11111
1

1111
1
1111
1  1
11111

11111
1  1
1

11111

1111
1   1

1   1
1111
```

✓

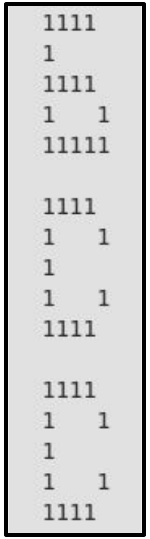
A



```
  11
 1 1
1   1
11111
1


  1
 1 1
1   1
11111
1   1


  1
 1 1
1   1
11111
1   1
```

✗

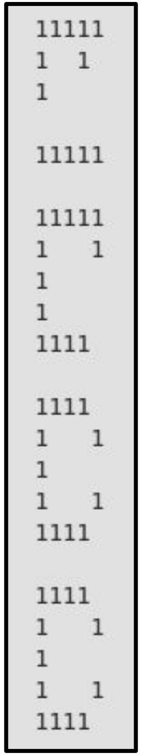
B



```
1111
1
1111
1   1
11111

1111
1   1
1
1   1
1111

1111
1   1
1
1   1
1111
```

✗

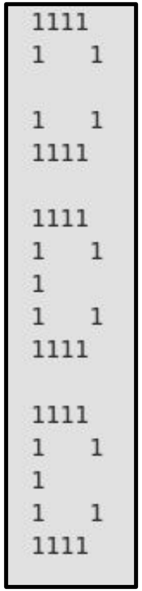
C



```
11111
1  1
1

11111

11111
1   1
1
1
1111

1111
1   1
1
1   1
1111

1111
1   1
1
1   1
1111
```

✗

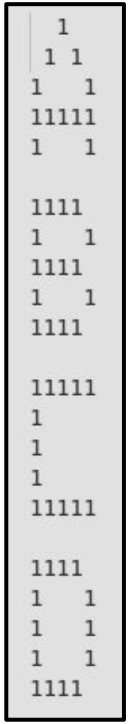
D



```
1111
1   1

1   1
1111

1111
1   1
1
1   1
1111

1111
1   1
1
1   1
1111
```

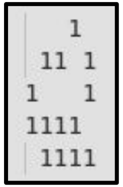
Testing

Conjunto A - B - C - D

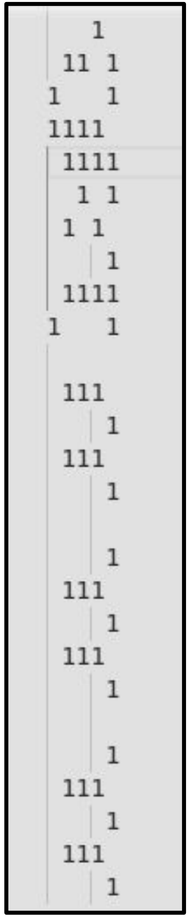


```
  1
 1 1
1   1
11111
1   1

1111
1   1
1111
1   1
1111

11111
1
1
1
11111

1111
1   1
1   1
1   1
1111
```

Mucho Ruido



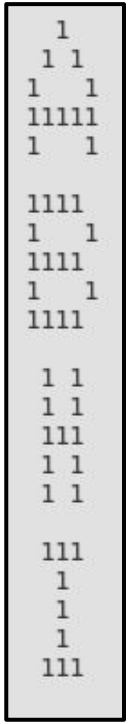
```
   1
 11 1
1   1
1111
 1111
```



```
   1
 11 1
1   1
1111
 1111
  1 1
 1 1
    1
 1111
1   1

 111
    1
 111
    1

    1
 111
    1
 111
    1

    1
 111
    1
 111
    1
```

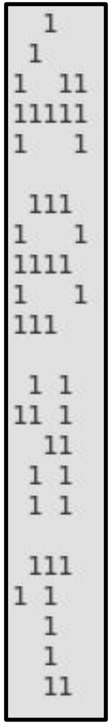
Testing

Conjunto A - B - H - I



```
  1
 1 1
1   1
11111
1   1

1111
1   1
1111
1   1
1111

 1 1
 1 1
 111
 1 1
 1 1

 111
  1
  1
  1
 111
```

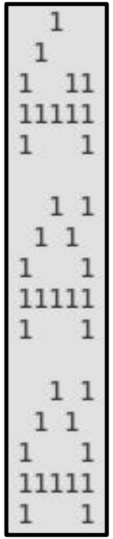
Un poco de ruido



```
  1
 1
1  11
11111
1   1

 111
1   1
1111
1   1
111

 1 1
11 1
  11
 1 1
 1 1

 111
1 1
  1
  1
  11
```

✗

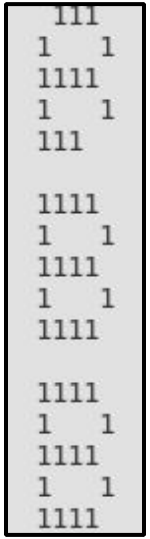
A



```
  1
 1
1  11
11111
1   1

  1 1
 1 1
1   1
11111
1   1

  1 1
 1 1
1   1
11111
1   1
```

✓

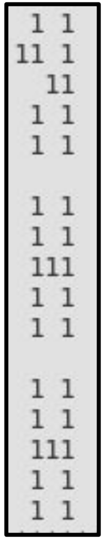
B



```
 111
1   1
1111
1   1
111

1111
1   1
1111
1   1
1111

1111
1   1
1111
1   1
1111
```

✓

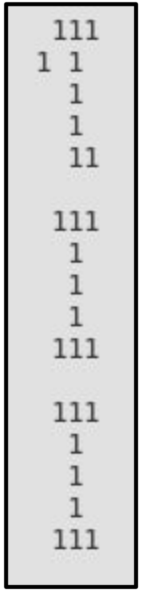
H



```
 1 1
11 1
  11
 1 1
 1 1

 1 1
 1 1
 111
 1 1
 1 1

 1 1
 1 1
 111
 1 1
 1 1
```

✓

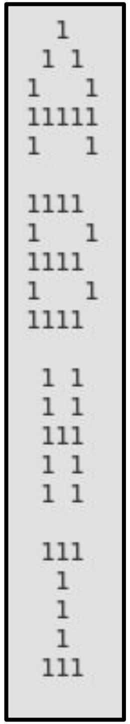
D



```
 111
1 1
  1
  1
  11

 111
  1
  1
  1
 111

 111
  1
  1
  1
 111
```

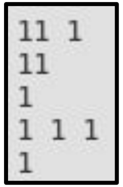
Testing

Conjunto A - B - H - I

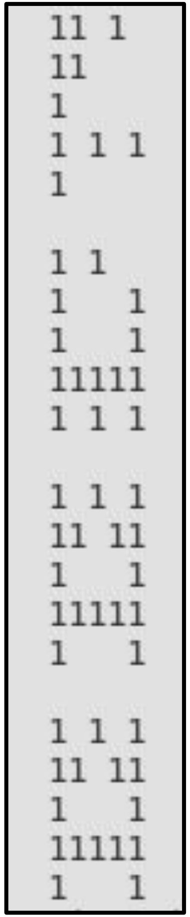


```
  1
 1 1
1   1
11111
1   1

1111
1   1
1111
1   1
1111

 1 1
 1 1
 111
 1 1
 1 1

 111
  1
  1
  1
 111
```

Mucho Ruido



```
11 1
11
1
1 1 1
1
```



```
11 1
11
1
1 1 1
1

1 1
1   1
1   1
11111
1 1 1

1 1 1
11 11
1   1
11111
1   1

1 1 1
11 11
1   1
11111
1   1
```