SVD_image_processing

May 26, 2019

1 SVD and image processing

We will look at how singular value decompositions can be applied to some problems in image processing. Be sure to evaluate the cell below before you begin.

In [1]:
```
from PIL import Image
import numpy as np
import csv

def display_image(matrix):
    return Image.fromarray(matrix.astype('uint8'))

def approximation(A, k):
    u, s, v = np.linalg.svd(A)
    sigma = np.zeros(A.shape)
    for i in range(k):
        sigma[i][i] = s[i]
    return np.dot(u, np.dot(sigma, v))

def square(ll, dims, fillcolor, strokecolor):
    x, y = ll
    w, h = dims
    points = [ll, [x+w,y], [x+w, y+h], [x, y+h]]
    return polygon(points, rgbcolor=fillcolor, axes=False) + polygon(points,rgbcolor=st

def display_vector(v):
    w = copy(v)
    w /= np.max(np.abs(v))
    return display_matrix(matrix(w).T)

def display_matrix(A):
    squares=[]
    y = A.nrows()-1
    for i, row in enumerate(A.rows()):
        x = 0
        for c in row:
            fillcolor = [c]*3
```

```
                if c < 0: fillcolor=[-c,0,0]
                strokecolor = [1 - f for f in fillcolor]
                squares.append(square([x,y], [1,1], fillcolor, strokecolor))
                x += 1
            y -= 1
        return sum(squares)
```

1.0.1 Image compression

Image files are typically pretty large so we would like to develop a way to store them more efficiently. A compression algorithm is a means of detecting redundancy in an image and storing only what is essential. For example, you may have seen the JPEG compression algorithm in your first linear algebra class. If you evaluate the cell below, it will load a matrix A, whose dimensions are 25×15 and whose entries are either 0 or 1.

```
In [32]: data = csv.reader(open('letter.csv'), delimiter= ' ')
         values = []
         for datum in data:
             datum.pop(-1)
             values.append([(1-float(d)) for d in datum])
         A = matrix(RDF, values)
         for row in values:
             print row
```

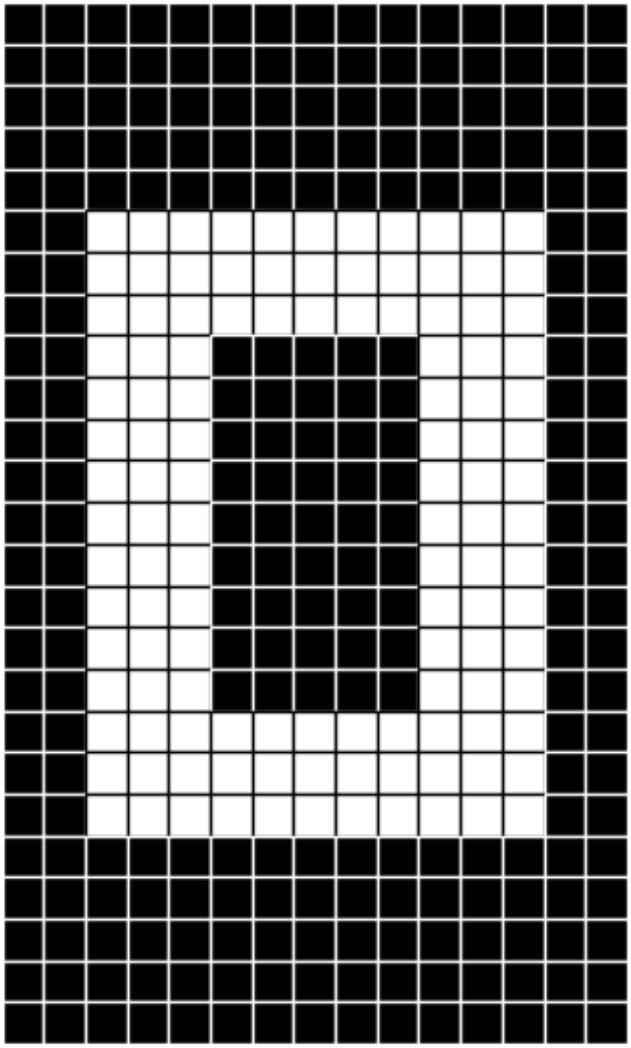
```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

If we interpret black as representing 0 and white as representing 1, this matrix represents an image as will be shown when you evaluate the cell below.

In [3]:
```
display_matrix(A)
```

Out[3]:



Remember that the rank of A is the dimension of the column space of A. What do you think the rank of this matrix is?

You can find the singular values of A below using `A.singular_values()`. Does this verify your intuition about the rank of A?

```
In [4]: A.singular_values()

Out[4]: [10.222386752739007,
         3.9373606740335614,
         1.6085005081556607e-15,
         8.393625040917807e-31,
         4.927645941373862e-32,
         6.920987624861464e-47,
         1.2552805804389235e-62,
         3.406889276165909e-64,
         1.612119600354043e-79,
         1.3025881243345454e-95,
         1.3311895002400113e-112,
         0.0,
         0.0,
         0.0,
         0.0]
```

Now find the singular value decomposition of A below.

```
In [31]: U, S, V = A.SVD()
```

You can pull out a column of a matrix `A` by `A.column(10)`, which will return a vector that is the 11th column of `A`. You can also display a vector using the `display_vector(v)` to see a visual representation of the vector. Once again, black represents 0, white represents 1, and red represents negative values.

How can you find a basis for the column space of A? Use the cell below to see a visual representation of the basis vectors.

```
In [39]: display_vector(U.column(1))

   Out[39]:
```