Character Recognition

Justin Wyss-Gallifent

July 21, 2021

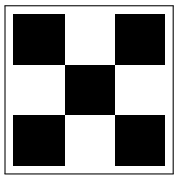
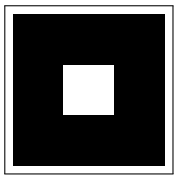
11.1 Introduction

The goal of this chapter is to introduce a very simple but reasonably effective method of recognizing characters. This can (and will) be implemented in a practical way (Matlab) but the real key is to understand what is going on mathematically.

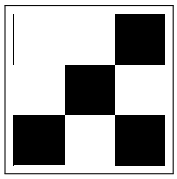
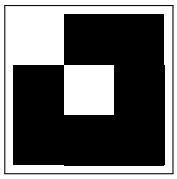
11.2 Simple Distance Checking

11.2.1 An Example

Before diving into complicated characters let's just look at the two characters X and O. To keep things really simple let's write each of them in a 3×3 grid:

Now consider these two, which we would intuitively accept as also X and O:

How can we say mathematically what we see intutively?

The obvious way might be to say that if we treat them as matrices where 0 indicates black and 1 indicates white then we have four matrices:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

What we'll actually do is unroll the matrices into vectors by simply putting each column underneath the previous one:

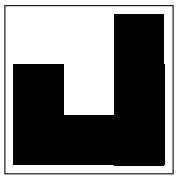
$$\bar{x} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \bar{x}_? = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \bar{o} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \bar{o}_? = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Then notice the following distances:

$$\begin{aligned} ||\bar{x} - \bar{x}_?|| &= 1 \\ ||\bar{x} - \bar{o}_?|| &= 2.4495 \\ ||\bar{o} - \bar{o}_?|| &= 1 \\ ||\bar{o} - \bar{x}_?|| &= 2.4495 \end{aligned}$$

Clearly and for obvious reasons the Xs are closer to each other than they are to the Os and the Os are closer to each other than they are to the Xs.

So what we could do would be really simplistic - given an unknown character we could simply ask how far it is from our X and from our O and decide appropriately. For example consider this:



If we convert this to a matrix and then unroll it to a vector:

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \bar{u} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Then we find:

$$||\bar{x} - \bar{u}|| = 2.2361$$
$$||\bar{o} - \bar{u}|| = 1.4142$$

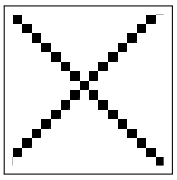
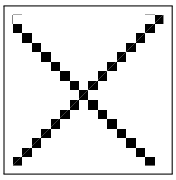
Since our unknown character is closer to an O, we call it an O. Voila!

Note: This could all have been done without unrolling the matrices but the unrolling will be necessary for what comes next.

11.2.2 There are Problems

There are many problems with this:

- Shifting a character slighly in a larger grid creates a huge distance. For example the following two Xs stored in 16×16 grids are distance 7.7460 apart when they are converted into vectors because one is shifted from the other in a way that misaligns the values.

- Rotating a character slightly does the same.
- There are many different versions of any character.

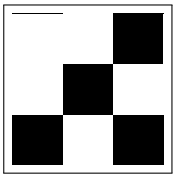
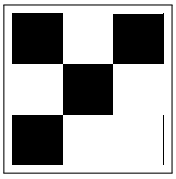
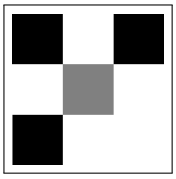
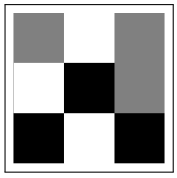
11.3 Developing a Robust SVD Method

11.3.1 Introduction

What we would really like to ask is more generally what a particular character looks like. In other words what are the mean features, what are more minor features, and so on.

11.3.2 The Essentials of a Character

Consider these four sort-of Xs. None is exactly an X but all of them would be accepted as such:

The vectors (unrolled matrices) corresponding to these are:

$$\bar{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \bar{x}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad \bar{x}_3 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0.5 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad \bar{x}_4 = \begin{bmatrix} 0.5 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0.5 \\ 0.5 \\ 0 \end{bmatrix}$$

If we put these together in a matrix and find the SVD:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0.5 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0.5 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0.5 \\ 1 & 1 & 1 & 0.5 \\ 0 & 1 & 1 & 0 \end{bmatrix} = U\Sigma V^T$$

where

$$U = \begin{bmatrix} -0.18 & -0.62 & 0.42 & -0.10 & 0 & -0.11 & 0.24 & -0.35 & 0.44 \\ -0.49 & -0.08 & -0.19 & 0.01 & 0 & -0.36 & 0.09 & -0.45 & -0.62 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ -0.49 & -0.08 & -0.19 & 0.01 & 0 & -0.35 & -0.63 & 0.28 & 0.36 \\ -0.07 & 0.19 & 0.03 & -0.98 & 0 & 0 & 0 & 0 & 0 \\ -0.49 & -0.08 & -0.19 & 0.01 & 0 & 0.84 & -0.05 & -0.10 & 0.03 \\ -0.05 & -0.17 & -0.68 & -0.05 & 0 & -0.13 & 0.59 & 0.27 & 0.23 \\ -0.43 & 0.09 & 0.49 & 0.06 & 0 & -0.03 & 0.35 & 0.62 & -0.21 \\ -0.26 & 0.72 & 0.07 & 0.16 & 0 & -0.11 & 0.24 & -0.35 & 0.44 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 4.1 & 0 & 0 & 0 \\ 0 & 1.3 & 0 & 0 \\ 0 & 0 & 0.57 & 0 \\ 0 & 0 & 0 & 0.35 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

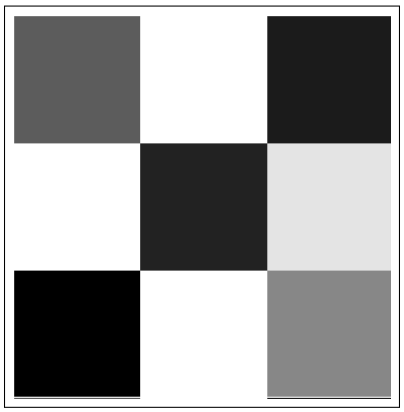
V = Omitted - Not Relevant

Since each column of M represents some variation on the character X, the vector $\bar{u}_1$ becomes the most important building block for all of those columns, meaning for all of those Xs. In other words that single column accounts for most of the variation in all the variations on the character X.

Subsequent columns $\bar{u}_2$, ... are less relevant, as indicated by the singular values.

In fact if we take the appropriate multiple of $\bar{u}_1$ such that the values are scaled between 0 and 1 we get:

$$\begin{bmatrix} 0.36 \\ 1 \\ 0 \\ 1 \\ 0.13 \\ 1 \\ 0.11 \\ 0.89 \\ 0.53 \end{bmatrix} \rightarrow \begin{bmatrix} 0.36 & 1 & 0.11 \\ 1 & 0.13 & 0.89 \\ 0 & 1 & 0.53 \end{bmatrix}$$



This is very clearly what we visually accept as an X.

Of course we may not want to only consider $\bar{u}_1$. If we were to take $\bar{u}_2$ into account we might suggest that:

$$\text{An X is mostly } \begin{bmatrix} -0.18 \\ -0.49 \\ 0 \\ -0.49 \\ -0.07 \\ -0.49 \\ -0.05 \\ -0.43 \\ -0.26 \end{bmatrix} \text{ with a bit of } \begin{bmatrix} -0.62 \\ -0.08 \\ 0 \\ -0.08 \\ 0.19 \\ -0.08 \\ -0.17 \\ 0.09 \\ 0.72 \end{bmatrix} \text{ thrown in.}$$

So perhaps when we're thinking about the building blocks for an X we might use both $\bar{u}_1$ and $\bar{u}_2$. We'll discuss how many of the $\bar{u}_i$ we might use a bit later.

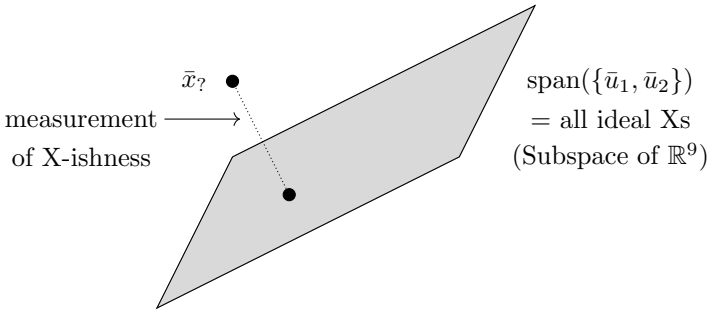
11.3.3 Comparing Another Character

Continuing the above example, suppose now we had another character and we wished to ask if we think it's an X.

We could suggest that if a (nonzero) character can be built out of $\bar{u}_1$ and $\bar{u}_2$ then it's definitely an X.

For a character we're testing it's almost certainly the case that we cannot build it exactly out of $\bar{u}_1$ and $\bar{u}_2$. So what we could ask instead if we can "almost" build this character out of $\bar{u}_1$ and $\bar{u}_2$.

A nice way to ask this in linear algebra would be to put this new character in a vector $\bar{x}_?$ and ask how far it is from the subspace spanned by $\bar{u}_1$ and $\bar{u}_2$. If it's not far, then it's almost an X. If it is far, then it's probably not an X.



(Note: All ideal Xs are not exactly the subspace spanned by $\bar{u}_1$ and $\bar{u}_2$ but rather the subset of that subspace in which all entries in the vectors are in the interval $[0, 1]$. However this alters nothing in the process so we generally gloss over it.)

In other words the distance from $\bar{x}_?$ to this subspace can be thought of as a measurement of the "X-ishness" of $\bar{x}_?$:

Since the vectors $\bar{u}_1$ and $\bar{u}_2$ form an orthonormal basis for the subspace they span, calculating this distance is easy.

For example if we take our actual, proper, original X (stored in the vector $\bar{x}$) from the beginning of the chapter we find:

$$\begin{aligned}
\text{dist}(\bar{x}, \text{span}(\{\bar{u}_1, \bar{u}_2\})) &= \left|\left|\bar{x} - \text{Proj}_{\text{span}(\{\bar{u}_1, \bar{u}_2\})}\bar{x}\right|\right| \\
&= ||\bar{x} - ((\bar{x} \cdot \bar{u}_1)\bar{u}_1 + (\bar{x} \cdot \bar{u}_2)\bar{u}_2)|| \\
&= 0.6340
\end{aligned}$$

Of course distance is relative but we might conclude that this is close enough.

On the other hand if we take our actual, proper, original O (stored in the vector $\bar{o}$) from the beginning of the chapter we find:

$$\begin{aligned}\text{dist}(\bar{o}, \text{span}(\{\bar{u}_1, \bar{u}_2\})) &= \left|\left|\bar{o} - \text{Proj}_{\text{span}(\{\bar{u}_1, \bar{u}_2\})}\bar{o}\right|\right| \\ &= ||\bar{o} - ((\bar{o} \cdot \bar{u}_1)\bar{u}_1 + (\bar{o} \cdot \bar{u}_2)\bar{u}_2)|| \\ &= 0.9790\end{aligned}$$

So certainly our original X is more of an X than our original O is.

11.3.4 Comprehensive SVD Summary

The general approach will be to construct a basis (called a *character basis*) for every character in our alphabet and then for any unknown character examine how close it is to the subspace spanned by each character basis and pick the one it's closest to.

More specifically for any given character α we take some number n of representative versions of α and construct the matrix for each. Each of these matrices gets unrolled into a vector in $\mathbb{R}^m$ where m is the number of pixels in each version. Call these vectors

$$\bar{\alpha}_1,\ \bar{\alpha}_2,\ ...,\ \bar{\alpha}_n$$

Place these vectors into a matrix and find the singular value decomposition:

$$[\bar{\alpha}_1 \quad \bar{\alpha}_2 \quad ... \quad \bar{\alpha}_n] = U\Sigma V^T$$

From the resulting U we pick a reasonable collection of vectors as determined by the singular values. Let's say we pick k of them, and we form a basis which encapsulates what it means to be that character. These vectors form the *character basis* for α. For convenience we put them in a matris called the *character basis matrix*:

$$B(\alpha) = [\bar{u}_1 \quad \bar{u}_2 \quad ... \quad \bar{u}_k]$$

This basis then spans the subspace $\text{col}(B(\alpha))$ called the *character subspace*.

Now then, for an unknown other character unrolled into a vector $\bar{x}$ we can measure its distance to α by measuring how close the vector is to $\text{col}(B(\alpha))$. This is essentially asking how easy it is to construct $\bar{x}$ (the character) out of the essential building blocks that make up α.

This distance is calculated by:

$$\text{dist}_\alpha \beta = \left|\left|\bar{x} - \text{Pr}_{\text{col}(B(\alpha))}\bar{x}\right|\right|$$

Luckily there's a shortcut for this when the matrix $B(\alpha)$ has orthonormal columns which it does in this case because our matrix $B(\alpha)$ was taken as columns of U:

$$\begin{aligned}
\text{Pr}_{\text{col}(B(\alpha))}\bar{x} &= (\bar{u}_1 \cdot \bar{x})\bar{u}_1 + ... + (\bar{u}_k \cdot \bar{x})\bar{u}_k \\
&= \bar{u}_1^T \bar{x}\bar{u}_1 + ... + \bar{u}_k^T \bar{x}\bar{u}_k \\
&= [\bar{u}_1 \ \ ... \ \ \bar{u}_k] \begin{bmatrix} \bar{u}_1^T \bar{x} \\ \vdots \\ \bar{u}_k^T \bar{x} \end{bmatrix} \\
&= [\bar{u}_1 \ \ ... \ \ \bar{u}_k] \begin{bmatrix} \bar{u}_1^T \\ \vdots \\ \bar{u}_k^T \end{bmatrix} \bar{x} \\
&= B(\alpha)B(\alpha)^T \bar{x}
\end{aligned}$$

(Note: Line 3 follows line 2 directly from the definition of $A\bar{x}$ as the linear combination of the columns of A using the weights in $\bar{x}$.)

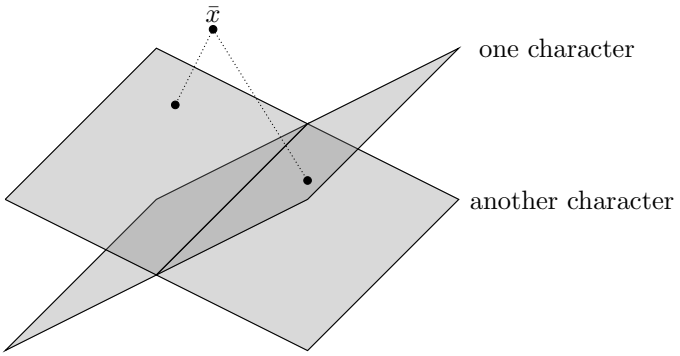
Consequently the distance between the unknown character vector and the known character basis is simply:

$$\left|\left|\bar{x} - B(\alpha)B(\alpha)^T \bar{x}\right|\right|$$

If we start the process with an entire collection of characters then an unknown character can be compared against each of them in turn and we can evaluate which one it's closest to.

More accurately we're taking each character basis for our known characters and seeing how close we can get to the unknown character using each character basis. The character basis that does the best job is the one we choose.

A crude visualization would be the following.



In reality there are many more subspaces, the dimension of those subspaces are higher and the dimension of the space they are in is also higher.

For example if we had 26 characters each using 16×16 resolution and if we used three columns of each U then we would have twenty-six (one per character) three-dimensional (basis for each) subspaces of $\mathbb{R}^{256}$ (256 pixels).

11.3.5 Choices

In the previous subsection we had to make two choices, the number of sample digits to use and the number of singular values to preserve. It's worth looking at this a bit more.

First, how many representative versions of a character should we take? It may be tempting to take as many as possible and as long as we don't take too many outliers (weird and esoteric version) then that's usually fine. If we use too many varied versions then the SVD gets confused about what's important in terms of variance.

Second, how many singular values should we preserve when we construct the character basis and character basis matrix? Again it may be temping to take more singular values but in reality this starts to cause problems. Why is this?

If our collection of characters is $m \times n$ (meaning we have n versions with m pixels each) then our U will be $m \times m$.

As we take more and more nonzero singular values, the character basis starts to span more and more of $\mathbb{R}^m$. As we do so, when we test a new character $\bar{d}$ the distance between $\bar{d}$ and $\text{span}(B(\alpha))$ starts to decrease. Consequently it becomes harder to compare a new character to a set of established characters.

Intuitively what's happening is that if we account for enough variation to draw every α then we have so many character basis vectors in A that every character (not just α) can potentially be built out of those character basis vectors. Consequently every character (not just α) begins to look like an α.

So figuring out how many singular values to preserve really comes down to

testing and seeing what actually works best when tested in the real world and double-checked by another approach, such as a human.

11.3.6 Barebones Summary and Partial Example

The barebones summary goes like this:

(a) For each character α, β, ... ,ω that we'd like to have in our database, take a bunch of sample characters, unroll them into vectors, put those vectors in a matrix, find the SVD and take the most significant columns of U. This gives us our character basis matrices $B(\alpha)$, $B(\beta)$, ..., $B(\omega)$.

(b) Take a character we'd like to identify, unroll it into a vector $\bar{x}$, then check the values:

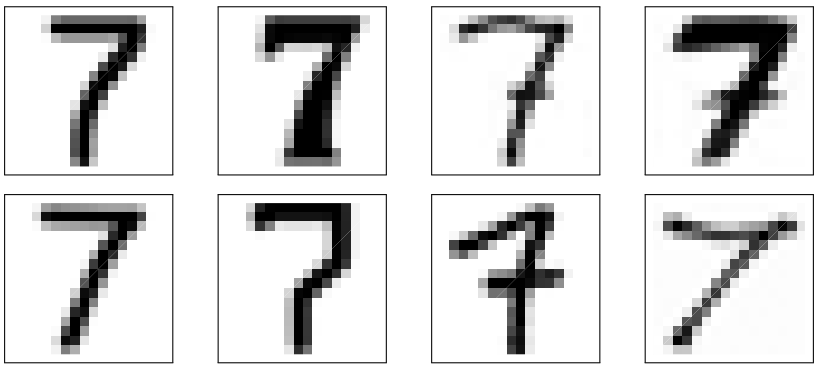
$$\begin{aligned}
\text{Distance from } \bar{x} \text{ to } \alpha &= \left|\left|\bar{x} - B(\alpha)B(\alpha)^T\bar{x}\right|\right| \\
\text{Distance from } \bar{x} \text{ to } \beta &= \left|\left|\bar{x} - B(\beta)B(\beta)^T\bar{x}\right|\right| \\
&\vdots = \vdots \\
\text{Distance from } \bar{x} \text{ to } \omega &= \left|\left|\bar{x} - B(\omega)B(\omega)^T\bar{x}\right|\right|
\end{aligned}$$

and choose the letter (α or β or ... or ω) corresponding to the smallest value.

Example 11.1. Suppose we wanted to do simple character recognition on the digits 0 through 9. We decide to use 16×16 resolution and we decide to preserve $k = 3$ columns of U.

First we would need to get some samples of each digit.

Here are eight different versions of the number 7 at a resolution of 16×16:



We convert each of these into a matrix, then unroll each to a vector (256 entries!), then put them all in a matrix (256×8) and find the SVD.

From the corresponding U we take the first $k = 3$ vectors and construct the character basis matrix:

$$B(7) = [\bar{u}_1 \;\; \bar{u}_2 \;\; \bar{u}_3]$$

This is a 256×3 matrix so we haven't explicitly written it out.

We would then repeat this for the digits 0,1,2,3,4,5,6,8,9 to get basis matrices $B(0)$, ..., $B(9)$ all together.

Now then, given a digit we'd like to identify we would unroll it into a vector $\bar{x}$ and check:

$$\begin{aligned}
\text{Distance from } \bar{x} \text{ to } 0 &= \left|\left|\bar{x} - B(0)B(0)^T\bar{x}\right|\right| \\
\text{Distance from } \bar{x} \text{ to } 1 &= \left|\left|\bar{x} - B(1)B(1)^T\bar{x}\right|\right| \\
&\vdots = \vdots \\
\text{Distance from } \bar{x} \text{ to } 9 &= \left|\left|\bar{x} - B(9)B(9)^T\bar{x}\right|\right|
\end{aligned}$$

and we would choose the digit (0 or 1 or ... or 9) with the smallest value.

As a side note for our 7 example above the singular values are

$$\{37.6566, 6.3112, 4.6349, 3.9108, 3.5626, 3.1247, 2.0267, 1.8722\}$$

So the first singular value captures most of the variance. If we take the corresponding $\bar{u}_1$ and multiply it by the appropriate scalar to get all values between 0 and 1 we get:

11.4 Comments

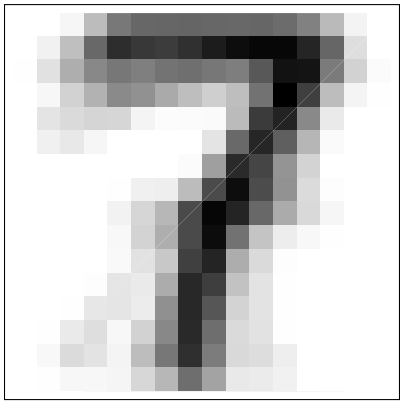
11.4.1 Visualizing the Basis

As we've commented earlier, the first column from U forms the fundamental building block for the character.

With the previous example here is $\bar{u}_1$ rolled into a matrix and scaled so all the values are between 0 and 1:

$$\begin{bmatrix}
1 & 1 & 0.96 & 0.74 & 0.47 & 0.41 & 0.4 & 0.39 & 0.4 & 0.41 & 0.4 & 0.42 & 0.51 & 0.73 & 0.95 & 1 \\
1 & 0.94 & 0.75 & 0.41 & 0.18 & 0.22 & 0.24 & 0.19 & 0.11 & 0.052 & 0.03 & 0.03 & 0.15 & 0.4 & 0.87 & 1 \\
0.99 & 0.88 & 0.68 & 0.53 & 0.47 & 0.5 & 0.45 & 0.44 & 0.47 & 0.5 & 0.34 & 0.07 & 0.079 & 0.48 & 0.82 & 0.98 \\
1 & 0.97 & 0.82 & 0.7 & 0.54 & 0.57 & 0.67 & 0.74 & 0.81 & 0.74 & 0.45 & 0 & 0.25 & 0.74 & 0.96 & 0.99 \\
1 & 0.89 & 0.85 & 0.83 & 0.84 & 0.95 & 0.99 & 0.98 & 0.98 & 0.73 & 0.23 & 0.14 & 0.54 & 0.92 & 1 & 1 \\
1 & 0.94 & 0.91 & 0.96 & 1 & 1 & 1 & 1 & 0.89 & 0.41 & 0.15 & 0.37 & 0.71 & 0.98 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0.98 & 0.62 & 0.17 & 0.27 & 0.58 & 0.83 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0.99 & 0.94 & 0.93 & 0.74 & 0.3 & 0.053 & 0.3 & 0.57 & 0.86 & 0.99 & 1 & 1 \\
1 & 1 & 1 & 1 & 0.95 & 0.84 & 0.72 & 0.29 & 0.024 & 0.14 & 0.41 & 0.67 & 0.85 & 0.96 & 1 & 1 \\
1 & 1 & 1 & 1 & 0.97 & 0.83 & 0.68 & 0.29 & 0.05 & 0.45 & 0.77 & 0.93 & 0.97 & 0.99 & 1 & 1 \\
1 & 1 & 1 & 1 & 0.98 & 0.89 & 0.78 & 0.25 & 0.14 & 0.68 & 0.86 & 0.99 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0.99 & 0.9 & 0.91 & 0.67 & 0.16 & 0.25 & 0.76 & 0.89 & 0.99 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0.9 & 0.9 & 0.92 & 0.56 & 0.16 & 0.34 & 0.83 & 0.89 & 0.99 & 1 & 1 & 1 & 1 \\
1 & 1 & 0.92 & 0.87 & 0.97 & 0.82 & 0.53 & 0.16 & 0.43 & 0.85 & 0.88 & 0.99 & 1 & 1 & 1 & 1 \\
1 & 0.97 & 0.86 & 0.89 & 0.95 & 0.74 & 0.47 & 0.19 & 0.49 & 0.85 & 0.86 & 0.93 & 1 & 1 & 1 & 1 \\
1 & 0.99 & 0.97 & 0.97 & 0.96 & 0.84 & 0.72 & 0.43 & 0.64 & 0.91 & 0.92 & 0.94 & 1 & 1 & 1 & 1
\end{bmatrix}$$

Here is is an image, we immediately see what we've got!



We might ask what the second and third columns of U look like but this question is much tricker. Because the first column of U is the primary building block it essentially adds the critical structure to the image, which is adding white where it needs to be (remember 0 is black and 1 is white, so we think of white as being added) and consequently we can scale the first column of U so that all values are nonnegative.

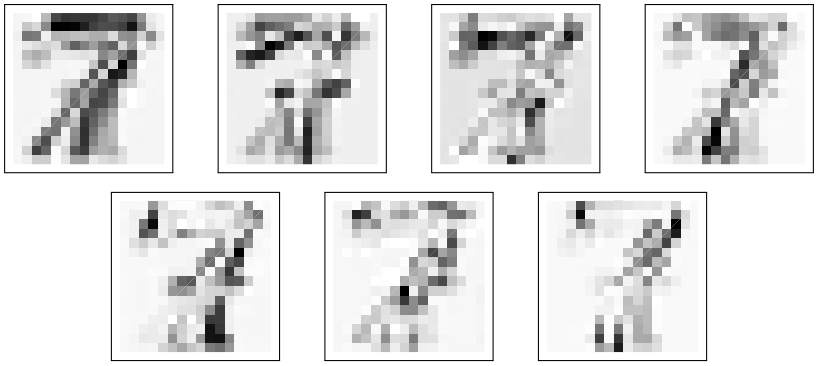
Subsequent columns add variation, however, and the values in those columns are usually a combination of positive and negative values.

For example here is $\bar{u}_2$ rolled into a matrix:

$$\begin{bmatrix}
0 & 0 & 0 & 0.05 & 0.12 & 0.13 & 0.13 & 0.13 & 0.13 & 0.12 & 0.11 & 0.11 & 0.12 & 0.07 & 0.01 & 0 \\
0 & -0.04 & -0.03 & 0.08 & 0.16 & 0.16 & 0.16 & 0.14 & 0.12 & 0.11 & 0.11 & 0.12 & 0.15 & 0.09 & 0.02 & 0 \\
-0.01 & -0.09 & -0.08 & -0.01 & 0.05 & 0.05 & 0.04 & 0.04 & 0.05 & 0.05 & 0.02 & 0.07 & 0.07 & -0.03 & -0.08 & -0.02 \\
0 & -0.01 & -0.02 & -0.03 & -0.02 & -0.06 & -0.1 & -0.08 & -0.06 & -0.09 & -0.1 & 0.03 & 0.07 & 0 & -0.03 & 0 \\
0 & -0.04 & -0.05 & -0.04 & -0.01 & -0.01 & -0.02 & -0.02 & -0.02 & -0.08 & -0.06 & 0.09 & 0.12 & 0.02 & 0 & 0 \\
0 & -0.03 & -0.04 & -0.02 & 0 & 0 & 0 & 0 & -0.04 & -0.11 & 0.03 & 0.16 & 0.09 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.02 & -0.11 & -0.03 & 0.15 & 0.14 & 0.05 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.03 & -0.03 & -0.1 & -0.06 & 0.11 & 0.12 & 0.05 & -0.03 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.02 & -0.05 & -0.09 & -0.06 & 0.1 & 0.12 & 0.1 & 0.01 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.05 & -0.1 & 0.06 & 0.12 & 0.12 & 0.09 & 0.02 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.02 & -0.1 & -0.03 & 0.12 & 0.11 & 0.08 & 0.06 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -0.01 & -0.09 & -0.07 & 0.04 & 0.13 & 0.1 & 0.07 & 0.04 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -0.08 & -0.09 & 0 & 0.07 & 0.13 & 0.09 & 0.06 & 0.04 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -0.08 & -0.11 & -0.03 & 0.01 & 0.08 & 0.13 & 0.08 & 0.05 & 0.05 & 0 & 0 & 0 & 0 & 0 \\
0 & -0.03 & -0.12 & -0.1 & 0 & 0.03 & 0.11 & 0.12 & 0.06 & 0.05 & 0.06 & 0.02 & 0 & 0 & 0 & 0 \\
0 & -0.01 & -0.03 & -0.03 & 0 & 0.02 & 0.06 & 0.06 & 0.01 & 0.02 & 0.03 & 0.02 & 0 & 0 & 0 & 0
\end{bmatrix}$$

In terms of building the 7s, the negative values here are were white is subtracted (or black is added) and the positive values here are where white is added (or black is subtracted) in order to get additional variation.

However it can be interesting to take the absolute values of these entries which gives a sense of where the most and least variation occurs after $\bar{u}_1$ has been taken into account. If we do this for $\bar{u}_2$ through $\bar{u}_8$ and then scale the values between 0 and 1 we get the following images:



11.4.2 Additonal Miscellaneous

In real world applications this produces reasonable but not acceptable results. There are several additional things that would be done if this were being actually implemented, including but not limited to:

- Cleaning images.
- Scaling images.
- Emphasizing black/white distinction.

- Centering characters.
- Rotating characters during testing.

11.5 Matlab

There are not a lot of new Matlab requirements for this section other than a few basic notes.

If we have an image stored in a matrix we can unroll it to a vector with the `reshape` command:

```
>> A = [1 4 7;2 5 8;3 6 9]
A =
     1     4     7
     2     5     8
     3     6     9
>> A = reshape(A,[9 1])
A =
     1
     2
     3
     4
     5
     6
     7
     8
     9
```

and then we can roll it back up again:

```
>> A = reshape(A,[3 3])
A =
     1     4     7
     2     5     8
     3     6     9
```

If we have a bunch of image matrices (already read and converted) we can put them together in a matrix easily. For example let's assume that $D1$ through $D9$ contain unrolled matrices for nine versions of the digit 7. To put these as columns into a matrix:

```
>> M = [D1 D2 D3 D4 D5 D6 D7 D8 D9];
```

Alternately you can add the columns to the matrix as you go:

```
>> M = [];
>> M = [M D1];
>> M = [M D2];
>> M = [M D3];
```

and so on...

To find the SVD and extract the first three columns:

```
>> [U,S,V] = svd(M);
>> B = U(:,1:3);
```

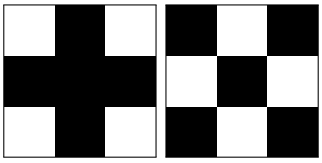
The notation here is that `:` takes all rows and `1:3` takes columns 1 through 3.

So now if we had another unrolled matrix `x` for a digit we could see how far it is from the above:

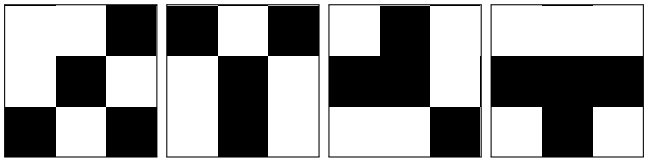
```
>> norm(x - B*transpose(B)*x)
```

11.6 Exercises

Exercise 11.1. Suppose your alphabet consists of the two 3×3 characters:

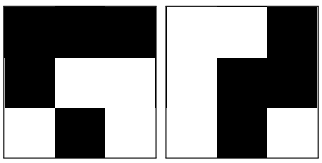


and you wish to identify each of the following characters:

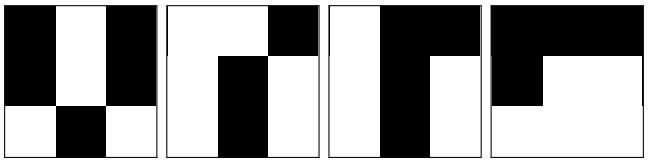


Convert these into matrices (white = 1 and black = 0) and unroll into vectors. Then identify each of the four characters using simple distance checking.

Exercise 11.2. Suppose your alphabet consists of the two 3×3 characters:

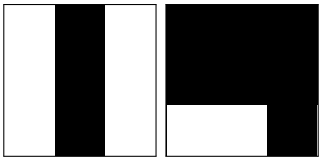


and you wish to identify each of the following characters:



Convert these into matrices (white = 1 and black = 0) and unroll into vectors. Then identify each of the four characters by using simple distance checking.

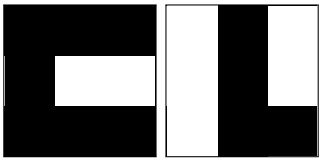
Exercise 11.3. Suppose your alphabet consists of the two 3×3 characters that look like a 1 and a 9 respectively:



(a) Design a character that looks like a 1 that will be recognized as a 9 using simple distance comparison. Show the calculations.

(b) Design a character that looks like a 9 that will be recognized as a 1 using simple distance comparison. Show the calculations.

Exercise 11.4. Suppose your alphabet consists of the two 3×3 characters that look like a C and an L respectively:

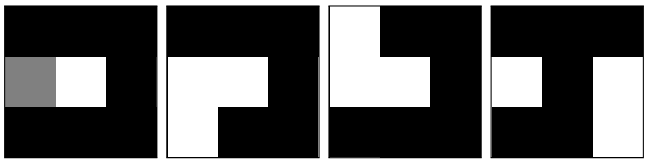


(a) Design a character that looks like a C that will be recognized as an L using simple distance comparison. Show the calculations.

(b) Design a character that looks like an L that will be recognized as a C using simple distance comparison. Show the calculations.

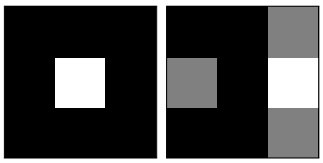
Exercise 11.5. Suppose you have the following four versions of the letter O:

And you have the following four versions of the letter J:



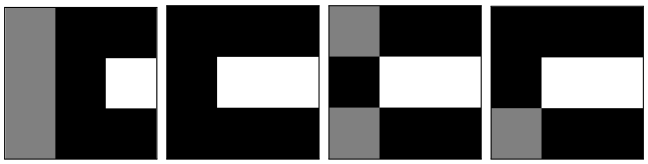
(a) Assuming values of 1.0 for white, 0.5 for gray and 0.0 for black, create letter basis matrices for O and for J using the two most important vectors from each associated U.

(b) Categorize the following two "unknown" characters by finding the distance between each character and the letter subspaces you constructed in (a).

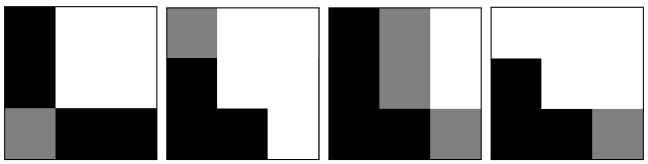


(c) For each letter basis matrix take the most important vector, multiply it by an appropriate scalar so all values lie between 0 and 1 with the largest value being 1, roll it back up to a matrix and plot. You are welcome to plot it by hand (reasonable shading) if you're not familiar enough with the Matlab commands. Do these look like an O and a J respectively?

Exercise 11.6. Suppose you have the following four versions of the letter C:

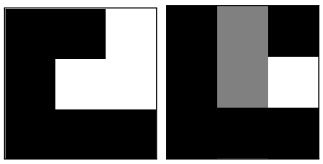


And you have the following four versions of the letter L:



(a) Assuming values of 1.0 for white, 0.5 for gray and 0.0 for black, create letter basis matrices for C and for L using the two most important vectors from each associated U.

(b) Categorize the following two "unknown" characters by finding the distance between each character and the letter subspaces you constructed in (a).



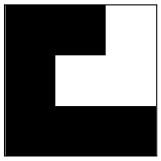
(c) For each letter basis matrix take the most important vector, multiply it by an appropriate scalar so all values lie between 0 and 1 with the largest value being 1, roll it back up to a matrix and plot. You are welcome to plot it by hand (reasonable shading) if you're not familiar enough with the Matlab commands. Do these look like an C and a L respectively?

Exercise 11.7. Suppose that in generating the basis matrix for one letter of your alphabet you took 256 versions of that letter with resolution 16×16 and just to be precise you took the entire of U as your letter basis matrix. Why would this be a bad idea?
Hint: Probably all the singular values will be distinct, why would this be a problem? Even if only most of them were, why would this be a problem?

Exercise 11.8. Suppose instead of using lots of different versions of the same letter to build a letter basis matrix you accidentally used the same version over and over.

(a) What would the letter basis matrix look like and why?

(b) If the letter looked like this 3×3 letter what would the leftmost vector in the letter basis matrix be?



Exercise 11.9. Suppose that your alphabet has two letters of size 2×2. When you build the letter basis matrices for them you find the following:

For the first letter when you do the SVD you find:

$$U = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & ... \\ \sqrt{2}/2 & \sqrt{2}/2 & ... \\ 0 & 0 & ... \\ 0 & 0 & ... \end{bmatrix}$$

For the second letter when you do the SVD you find:

$$U = \begin{bmatrix} \sqrt{3}/3 & \sqrt{2}/2 & ... \\ 0 & 0 & ... \\ \sqrt{3}/3 & -\sqrt{2}/2 & ... \\ \sqrt{3}/3 & 0 & ... \end{bmatrix}$$

Identify this letter as either the first letter or second letter: