Pattern Recognition

Lecture 2

Nearest Neighbors

Thomas M. Breuel

Homework

How well did it work?

How can you improve it?

Similarity

input vectors are vectors in R^n

actually in $\{0,1\}^n$

you used

```
sqrt(sum((x-y)**2))
```

Euclidean distance

- why?
- what are its properties?

Distance Metric

A *distance metric* is a function d: S x S → R over a set S satisfying

- $d(x, y) \geq 0$ — non-negativity
- $d(x, y) = 0$ iff x=y — identity of indescernibles
- $d(x, y) = d(y, x)$ — symmetry
- $d(x, z) \leq d(x, y) + d(y, z)$ — triangle inequality

Norm

A norm is a function p: V → R on a vector space V satisfying

$p(a \cdot v) = |a| \, p(v)$ — positive homogeneity

$p(u+v) \leq p(u) + p(v)$ — triangle inequality

$p(v) = 0$ iff v is the zero vector — positive definiteness

on a normed vector space

$d(x, y) = \|x - y\|$ defines a distance

$\|x\| = d(x, 0)$ defines a norm

Standard Vector Space Norms

$\|x\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$ — Minkowski norm

special cases:

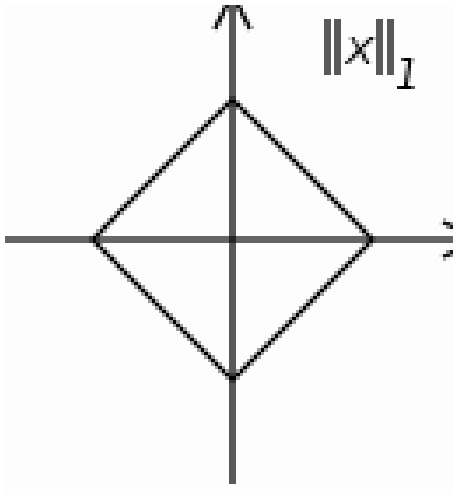
$\|x\|_2 = \sqrt{\sum_i x_i^2}$ — Euclidean norm

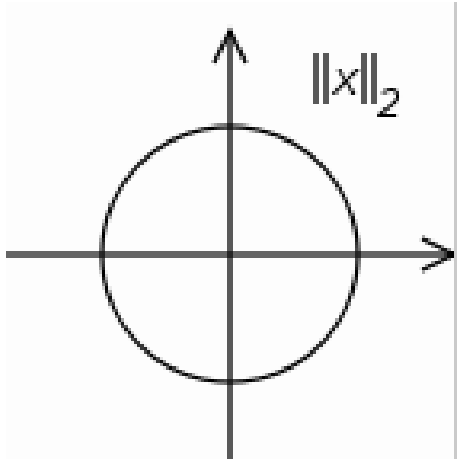
$\|x\|_1 = \sum_i |x_i|$ — Manhattan norm, Taxicab norm

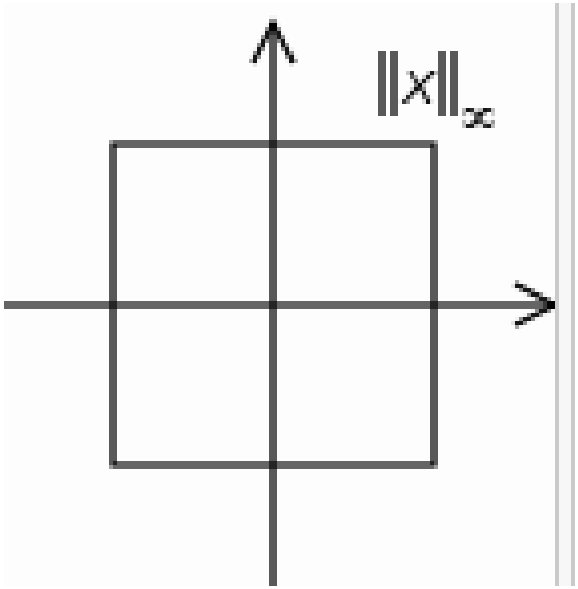
$\|x\|_0 = \sum_i [x_i \neq 0]$ — Hamming distance

$\|x\|_\infty = max|x_i|$ — maximum norm

Unit Balls







Euclidean Norm & Inner Product

$$\|x\|_2 = \sqrt{x \cdot x}$$

$$d(x, y)^2 = (x - y) \cdot (x - y) = x \cdot x - 2\, x \cdot y + y \cdot y = \|x\|^2 + \|y\|^2 - 2(x \cdot y)$$

```
def all_dists(M,y):
   # M's rows are the xs
   xs2 = sum(M**2,axis=1)
   y2 = sum(y**2)
   return xs2 + y2 – 2 * dot(M,x)
```

Remember

Distance metrics need to satisfy the triangle inequality.

Everything else is called a *dissimilarity measure*.

dissimilarity measure

- 0 = minimum dissimilarity
- bigger for more dissimilarity

similarity measure

0 = minimum similarity

bigger for more similarity

Other (?) Dissimilarity Measures

- **Bray-Curtis** $\sum |u_i - v_i| / \sum |u_i + v_i|.$
- **Canberra** $\frac{\sum_i |u_i - v_i|}{\sum_i |u_i| + |v_i|}.$
- **Chebyshev** $\max_i |u_i - v_i|.$
- **Cityblock** $\Sigma_i \left| u_i - v_i \right|$
- **Correlation** $\frac{1 - (u - \bar{u})(v - \bar{v})^T}{||(u - \bar{u})||_2 ||(v - \bar{v})||_2^T}$
- **Cosine** $\frac{1 - uv^T}{||u||_2 ||v||_2}.$
- **Minkowski** $||u - v||_p = (\sum |u_i - v_i|^p)^{1/p}.$

Dissimilarities for Binary Vectors

$$\delta(A,B)=|A\cup B|-|A\cap B|$$

- Dice

$$D(A,B)=\frac{\delta(A,B)}{|A\cup B|+|A\cap B|}$$

- Hamming / Matching

$$H(A,B)=\frac{\delta(A,B)}{n}$$

- Jaccard

$$J(A,B)=1-\frac{|A\cap B|}{|A\cup B|}$$

- Kulsinski

$$K(A,B)=1-\frac{|A\cap B|}{\delta(A,B)+n}$$

Distances and Dissimilarity Measures

Why are there so many?

Which ones work best for our data?

How can they be justified?

Which properties are important?

- symmetry?
- indiscernibles?
- triangle inequality?

Homework: try to answer these questions empirically for MNIST

Implementations of Dissimilarity Measures

`scipy.spatial.distance` contains a lot of dissimilarity measures

slower:

```
ds = [distance.euclidean(x,v) for x in protos]
best = argmin(ds)
```

faster:

```
ds = distance.cdist(v[newaxis,:],prototypes,'euclidean')
best = argmin(ds)
```

Input Vector Scale

Input vectors are vectors in R^n

The scale of the individual entries is arbitrary (it represents an intensity measurement in some arbitrary units):

$x_i \in [0,1]$
$x_i \in [0,255]$

(Watch out: mnist_read reads values from 0..255)

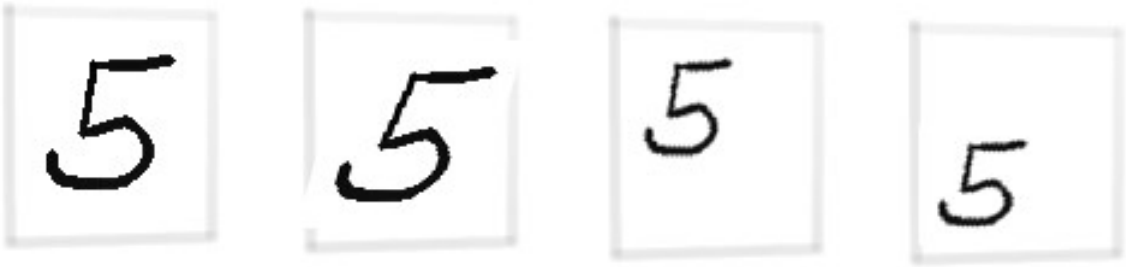
Does the nearest neighbor classifier depend on a change of scale?

$$d(x, y) < d(x, z) \quad \textit{iff} \quad d(a \cdot x, a \cdot y) < d(a \cdot x, a \cdot x)$$

For which dissimilarity measures is that true?

Can you prove it?

Pattern Transformations



often, there are invariances in pattern recognition

e.g., character identity does not depend on position

(but sometimes it does, like “p” vs. “P”)

how can we account for this?

Idea 1: Geometric Matching

The original pattern u has undergone some transformation T resulting in the vector $v = T \cdot x$

When classifying v by nearest neighbor, there should be an inverse transformation T^{-1} corresponding to the original T

Then, $d(u, T^{-1} v) = d(u, T^{-1} T u) = d(u, u) = 0$

Since we don't know T, we try all of them and define a new dissimilarity measure d'

$$d'(u, v) = arg\,min_T\, d(u, T v)$$

Direct Matching under Translations

for the unknown pattern v , shift it around at all possible translations

compare each shifted version with each prototype

return the prototype with the minimum distance to some shifted version of the input pattern

Complexity:

prototypes x # shifts

50000 x 28 x 28

too big...

Idea 2: Centroid Matching

General idea:

compute the center of each pattern

shift the prototype relative to the pattern so that their centers align

then compute the distance as before

Centroid as the Pattern Center

What is the center of a pattern?

- “average” x coordinate
- “average” y coordinate

This is called the *centroid*

Centroid Computation

assume all pixels I_{ij} are either 0 or 1; then sum up all the x and y coordinates where the image $I_{i,j}$ is not zero and divide by the total number of non-zero pixels

$$\bar{x} = \frac{\sum_{x,y} x \cdot I_{x,y}}{\sum_{x,y} I_{x,y}}$$

$$\bar{y} = \frac{\sum_{x,y} y \cdot I_{x,y}}{\sum_{x,y} I_{x,y}}$$

This also works for fractional $I_{i,j}$

Centroid Computation

Rough idea:

```
xs = ones(w)[:,newaxis] * arange(h)[newaxis,:]
xc = sum(xs * I) / sum(I)
ys = arange(w)[:,newaxis] * ones(h)[newaxis,:]
yc = sum(ys * I) / sum(I)
```

Remember, though, that Python reverses the coordinates, so “xs” is actually y coordinates in the image.

Comparison under Centroids

We can define a new dissimilarity measure based on centroids:

$$d'(u, v) = d(u, T_\delta v)$$

where

$$T_\delta = C(u) - C(v)$$

and $C(u)$ is the centroid of u as defined above

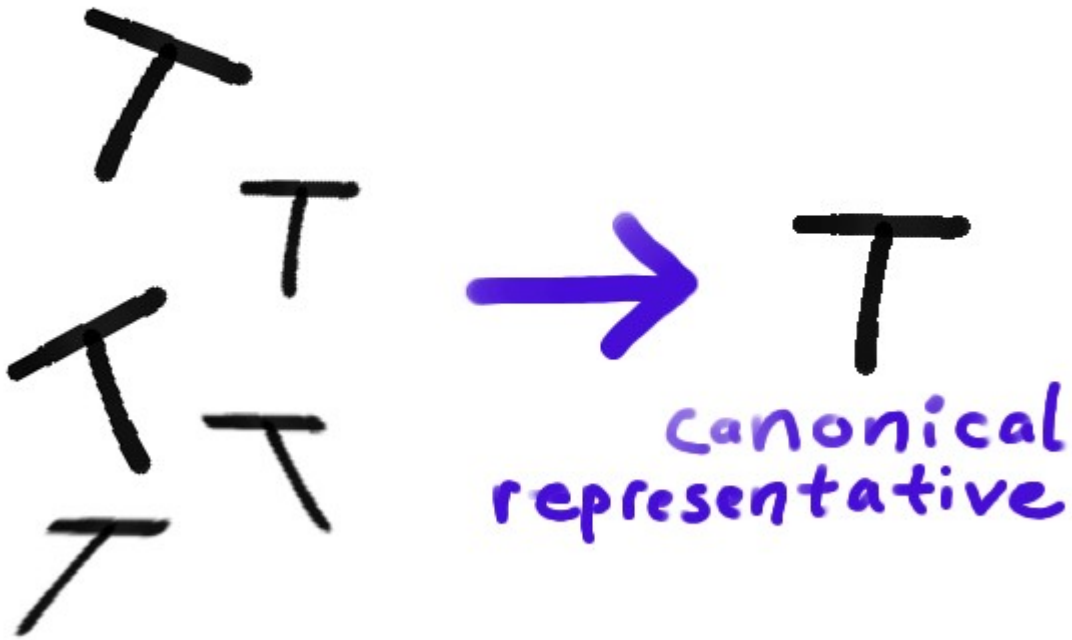
Centroids vs Matching

Think about...

How does comparison under matching centroids compare to geometric matching (= trying all translations)?

Idea 3: Canonicalization

Instead of defining a different dissimilarity measure d', we can transform the patterns themselves into a *canonical representative*.



Canonicalization under Translation

input image v is 28 x 28 image

compute the centroid $C(v)$ for each pattern

shift the input image so that the centroid is always at the same location

- mathematically, we choose $c=(0,0)$, the origin
- in programming, we choose $c=(w/h\,,h/2)$, the center of the image

the canonicalization transformation $f(v)$ is

$$v'=f(v)=T_{c-C(v)}v$$

Invariant Representation

the canonicalized vector v' is an example of a *translation-invariant representation*

i.e., for any translation T, we have that

$$f(v) = f(T\,v)$$

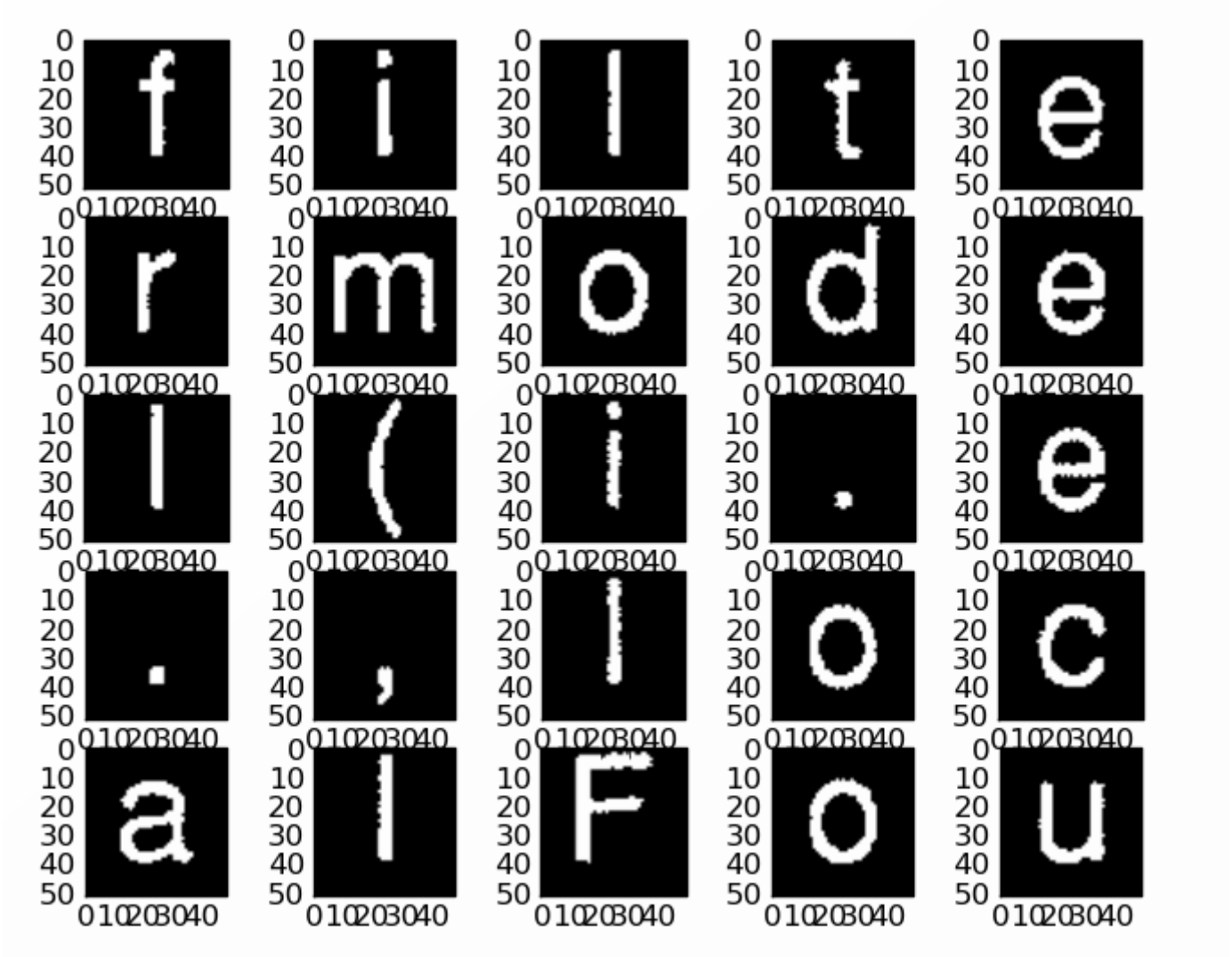
Canonicalization for Translation + Size by Bounding Box

observation: absolute size of character also doesn't matter for classifying the character

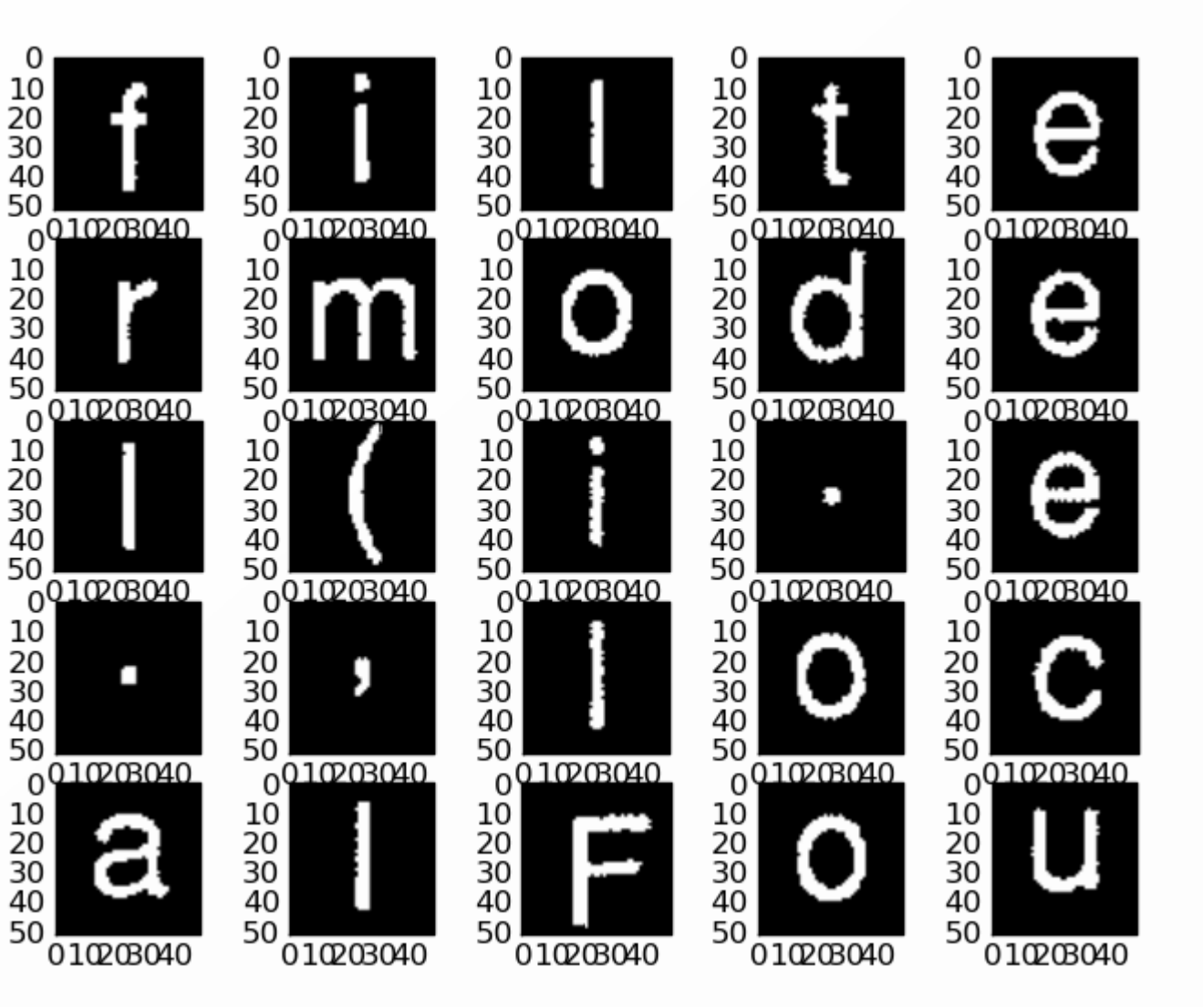
idea: compute the bounding box of the input character, then...

- rescale the bounding box to fit into a square of a given target size
- isotropically rescale the bounding box to fit into a square of a given target size; center the smaller dimension
- isotropically rescale, but don't magnify

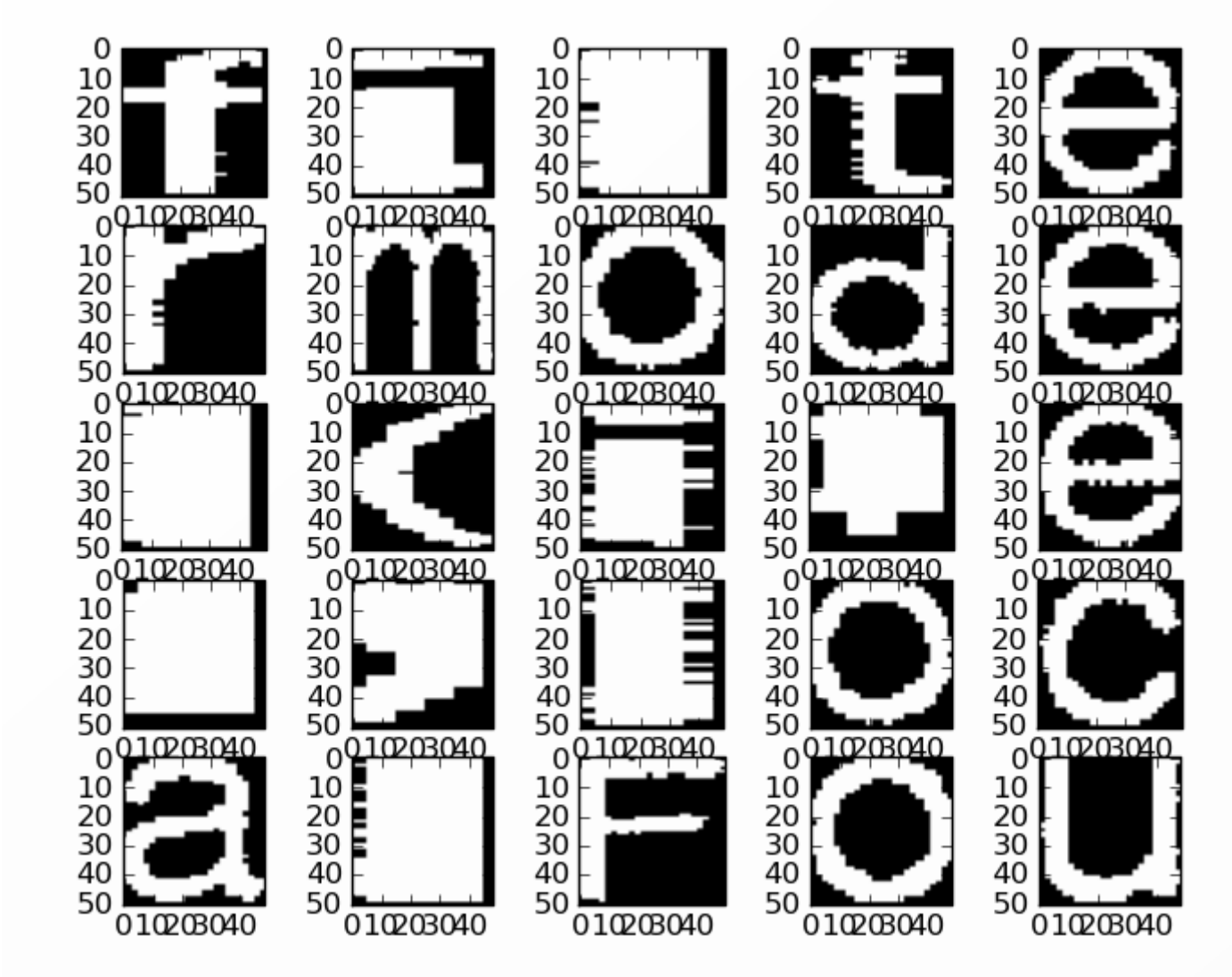
Uncentered Characters



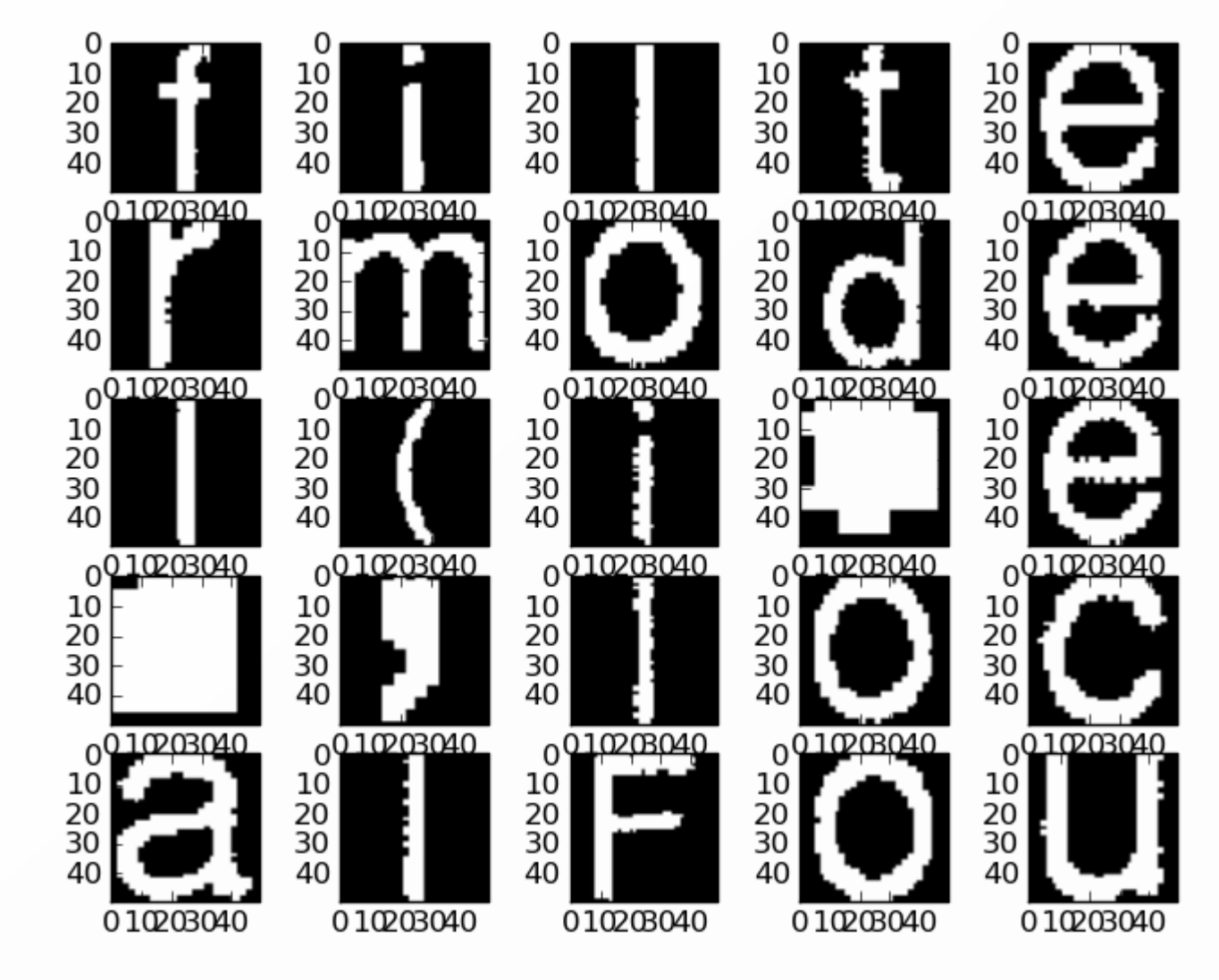
Normalized by Centroid



Anisotropic Size Normalization



Isotropic Size Normalization



Isotropic Size Normalization with Scale LImit