Programming Project 2 Problem #1

Michael Foster

MA 427

This problem involves some standard techniques in machine learning and data science. In particular we will: use a large-ish data set, load the data, check that everything looks OK, reformat it, do a quick visual check, then use the training data to create a model and then use test data to check how well the model works.

1.

a.

```
clear all
load mnist_all
who
```

```
Your variables are:

test0    test1    test2    test3    test4    test5    test6    test7    test8    test9    train0   train1   train2   tr
```

```
size test0
```

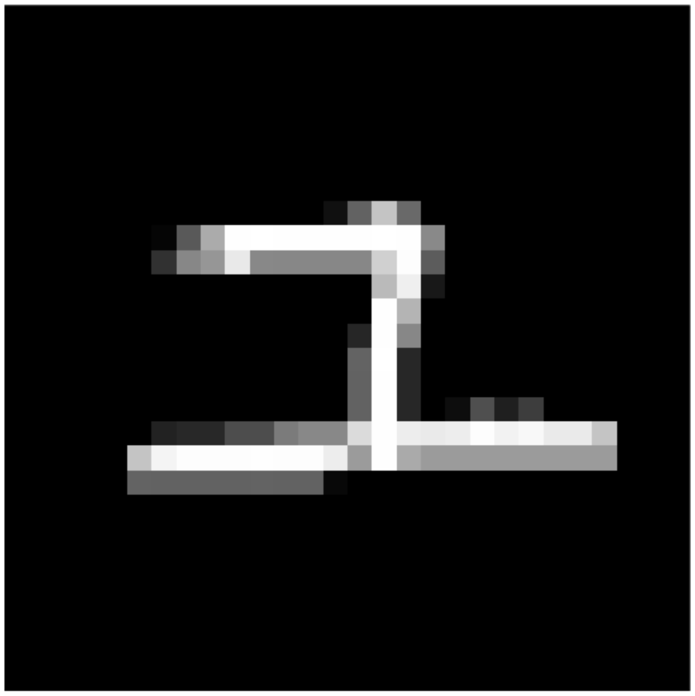
```
ans = 1×2
     1     5
```

```
size train0
```

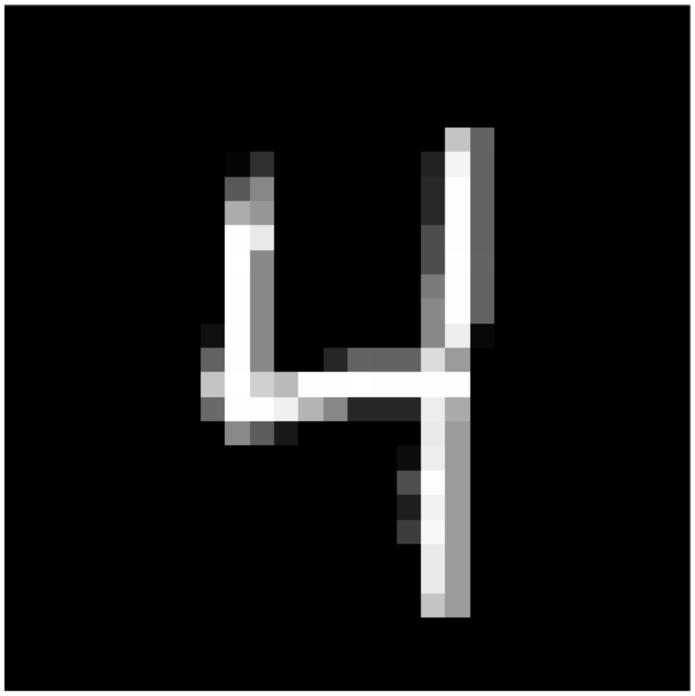
```
ans = 1×2
     1     6
```

b.

```
digit = train4(11,:);
digit = reshape(digit,28,28);
image(digit)
colormap(gray(256))
axis square tight off
```

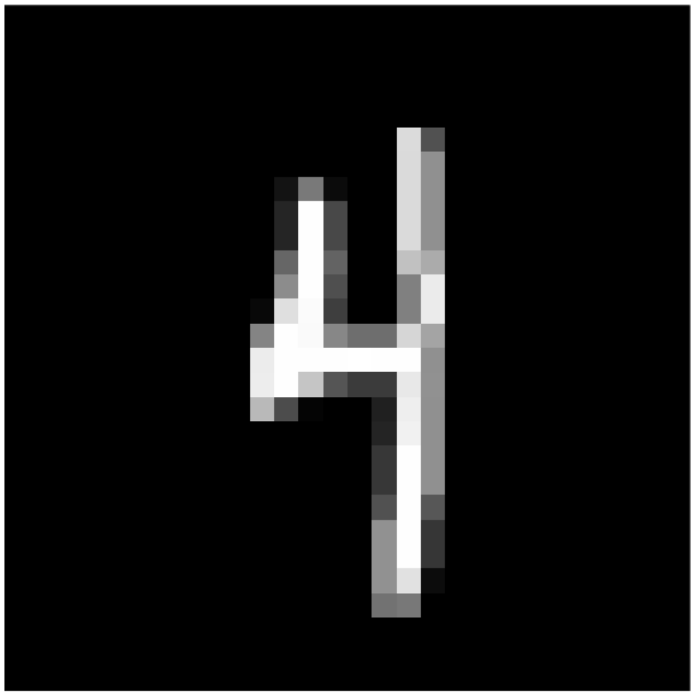


```
image(flip(rot90(digit)))
axis square tight off
```

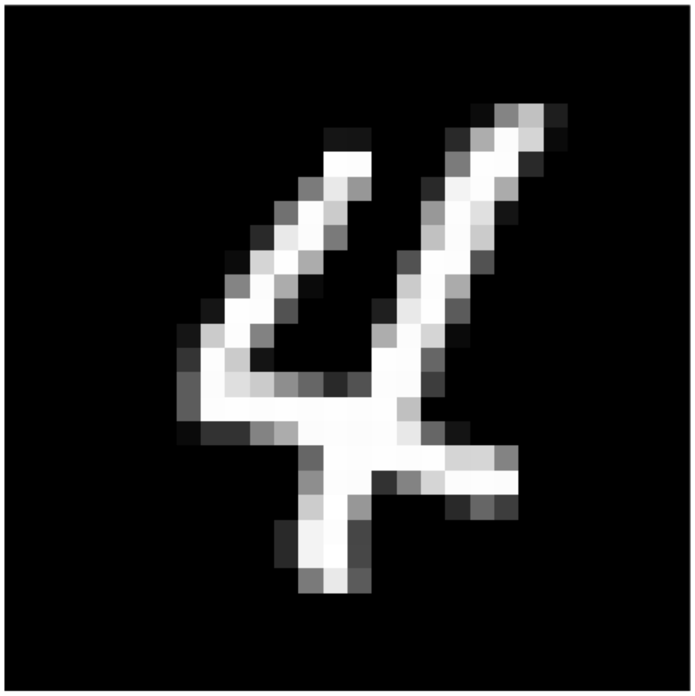


c.

```
showdigit(test4,11)
```

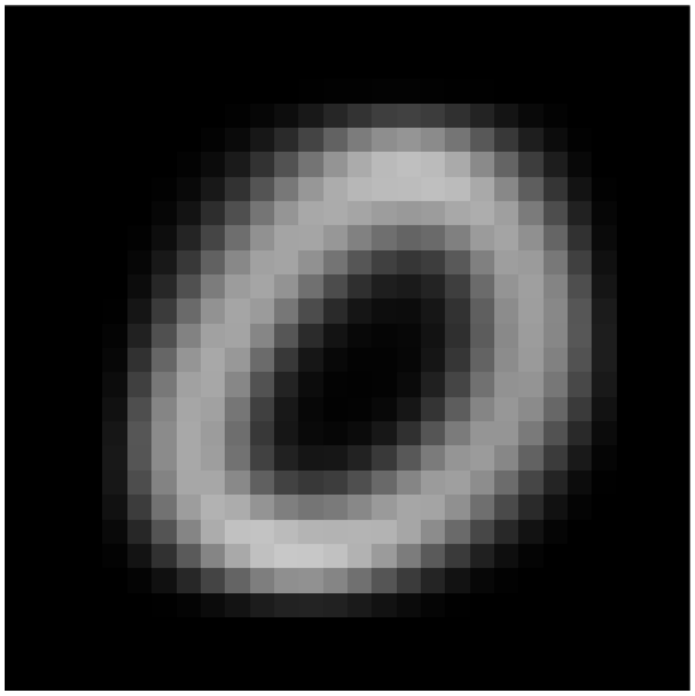


```
showdigit(test4,50)
```

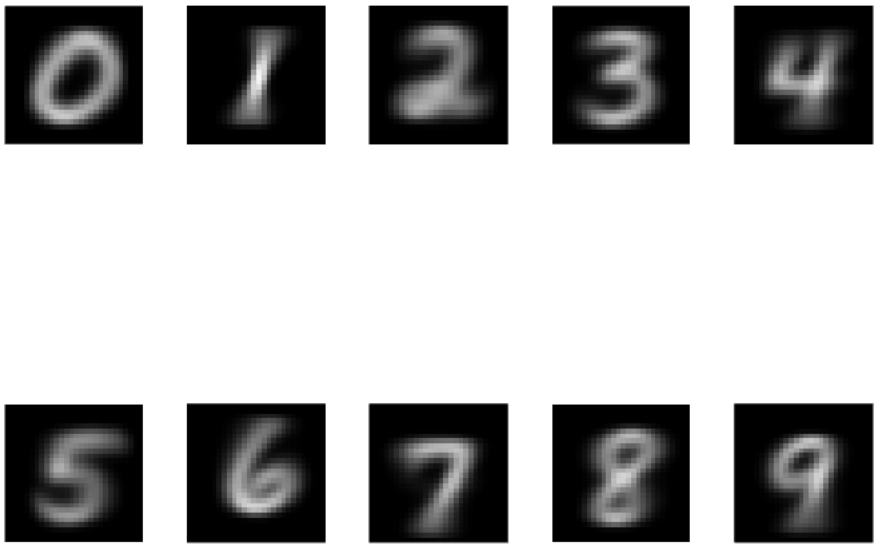


d.

```
AvgLet = zeros(10,784);
AvgLet(1,:) = mean(train0);
AvgLet(2,:) = mean(train1);
AvgLet(3,:) = mean(train2);
AvgLet(4,:) = mean(train3);
AvgLet(5,:) = mean(train4);
AvgLet(6,:) = mean(train5);
AvgLet(7,:) = mean(train6);
AvgLet(8,:) = mean(train7);
AvgLet(9,:) = mean(train8);
AvgLet(10,:) = mean(train9);
showdigit(AvgLet,1)
```



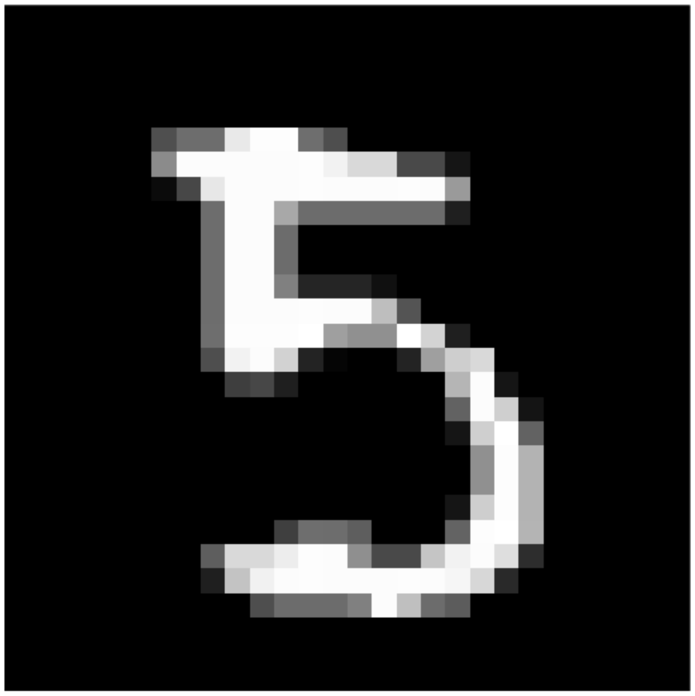
```
for i = 1:10
    subplot(2,5,i)
    showdigit(AvgLet,i)
end
```



```
clf
```

e.

```
digit = test5(73,:);
showdigit(test5,73)
```



```
dist = double(digit)-AvgLet;
norms = sqrt(sum(dist.^2,2));
norms'
```

```
ans = 1×10
10^3 ×
    2.1577    2.3412    2.2587    1.7699    2.1490    1.7597    2.2314    2.1840 ···
```

```
[~,idx] = min(norms);
idx-1
```

```
ans = 5
```

f.

```
match(test5(73,:),AvgLet)
```

```
ans = 5
```

g.

```
rng(20241206)
for k = 0:9
    mytest = eval(['test',num2str(k)]);
    randrow = randi(size(mytest,1));
    y = match(mytest(randrow,:),AvgLet);
    fprintf('test digit %d is closest to average digit %d\n',k,y)
end
```

```
test digit 0 is closest to average digit 0
test digit 1 is closest to average digit 1
test digit 2 is closest to average digit 2
test digit 3 is closest to average digit 3
test digit 4 is closest to average digit 4
test digit 5 is closest to average digit 5
test digit 6 is closest to average digit 6
test digit 7 is closest to average digit 7
test digit 8 is closest to average digit 9
test digit 9 is closest to average digit 9
```

The code worked almost perfectly—the only trip-up was that it matched 8 to 9. It missed 1 digit.

h.

```
hits = 0;
for row = 1:size(test0,1)
    y = match(test0(row,:),AvgLet);
    if y == 0
        hits = hits+1;
    end
end
fprintf("Correct match %.2f%% of the time \n", hits/size(test0,1)*100)
```

```
Correct match 89.59% of the time
```

The accuracy rate is 89.59%, so the failure rate is 10.41%. I'd say that's pretty decent!

j.

Digit 1:

```
hits = 0;
for row = 1:size(test1,1)
    y = match(test1(row,:),AvgLet);
    if y == 1
        hits = hits+1;
    end
end
fprintf("Correct match %.2f%% of the time \n", hits/size(test0,1)*100)
```

```
Correct match 111.43% of the time
```

Digit 1 has a failure rate of -11.43%? This could be over 100% because the singular line in the digit 1 could be faintly recognized in other digits like 9 and 4 and therefore more samples are taken to be the digit 1?

Digit 2:

```
hits = 0;
for row = 1:size(test2,1)
```

```
    y = match(test2(row,:),AvgLet);
    if y == 2
        hits = hits+1;
    end
end
fprintf("Correct match %.2f%% of the time \n", hits/size(test0,1)*100)
```

```
Correct match 79.69% of the time
```

Digit 2 has a failure rate of 20.31%

Digit 3:

```
hits = 0;
for row = 1:size(test3,1)
    y = match(test3(row,:),AvgLet);
    if y == 3
        hits = hits+1;
    end
end
fprintf("Correct match %.2f%% of the time \n", hits/size(test0,1)*100)
```

```
Correct match 83.06% of the time
```

Digit 3 has a failure rate of 16.94%

Digit 4:

```
hits = 0;
for row = 1:size(test4,1)
    y = match(test4(row,:),AvgLet);
    if y == 4
        hits = hits+1;
    end
end
fprintf("Correct match %.2f%% of the time \n", hits/size(test0,1)*100)
```

```
Correct match 82.76% of the time
```

Digit 4 has a failure rate of 17.24%

Digit 5:

```
hits = 0;
for row = 1:size(test5,1)
    y = match(test5(row,:),AvgLet);
    if y == 5
        hits = hits+1;
    end
end
fprintf("Correct match %.2f%% of the time \n", hits/size(test0,1)*100)
```

```
Correct match 62.45% of the time
```

Digit 5 has the highest failure rate at 37.55%. I think this could be because I feel like this digit has the most room for error when writing it—it is kind of an awkward number to write so most likely the model took the 5's to be other numbers such as 3. From first glance at the y-values I computed, there are lots of 3's, 4's, and 6's.

Digit 6:

```
hits = 0;
for row = 1:size(test6,1)
    y = match(test6(row,:),AvgLet);
    if y == 6
        hits = hits+1;
    end
end
fprintf("Correct match %.2f%% of the time \n", hits/size(test0,1)*100)
```

```
Correct match 84.39% of the time
```

Digit 6 has a failure rate of 15.61%

Digit 7:

```
hits = 0;
for row = 1:size(test7,1)
    y = match(test7(row,:),AvgLet);
    if y == 7
        hits = hits+1;
    end
end
fprintf("Correct match %.2f%% of the time \n", hits/size(test0,1)*100)
```

```
Correct match 87.35% of the time
```

Digit 7 has a failure rate of 12.65%

Digit 8:

```
hits = 0;
for row = 1:size(test8,1)
    y = match(test8(row,:),AvgLet);
    if y == 8
        hits = hits+1;
    end
end
fprintf("Correct match %.2f%% of the time \n", hits/size(test0,1)*100)
```

```
Correct match 73.27% of the time
```

Digit 8 has a failure rate of 26.73%. The same theory as digit 5 could be applied here.

Digit 9:

```
hits = 0;
for row = 1:size(test9,1)
```

```
    y = match(test9(row,:),AvgLet);
    if y == 9
        hits = hits+1;
    end
end
fprintf("Correct match %.2f%% of the time \n", hits/size(test0,1)*100)
```

```
Correct match 83.06% of the time
```

Digit 9 has a failure rate of 16.94%

c.

```
function [] = showdigit(data, row)
digit_fun = data(row, :);
digit_fun = reshape(digit_fun,28,28);
image(flip(rot90(digit_fun)))
colormap(gray(256))
axis square tight off
end
```

f.

```
function y = match(digit,T)
dist = double(digit) - T;
norms = sqrt(sum(dist.^2,2));
[~,idx] = min(norms);
y = idx-1;
end
```