Ross Lewis

425 - Machine Learning – Homework 4

PART 1

Problem I:

I tried three models. The first has 1 hidden layer, uses relu as an activation function, and has a softmax output. The second has 2 hidden layers and uses sigmoid activation and output functions. The last has 3 hidden layers, uses relu as an activation function, and softmax as an output function.

```
#Model 1
#1 hidden layer
#relu activation
#softmax output
model1 = Sequential()
model1.add(Dense(128, activation='relu', input_shape=(784,)))
model1.add(Dense(num_classes, activation='softmax'))

model1.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
print(model1.summary())

model1.fit(xtrain, ytrain,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1)

ypred = model1.predict_classes(xval)

conf, acc, rec, prec = func_confusion_matrix(yval,ypred)
print(conf)
print(acc)
print(rec)
print(prec)
```

```
[[ 951    0    1    2    1    0    0    2    3    0]
 [   0 1035    5    0    1    0    0    3    8    2]
 [   1    2  987    6    1    0    2    7    3    1]
 [   1    0    9 1001    2    9    0    5    3    2]
 [   2    0    3    0 1017    0    4    4    1   11]
 [   0    1    1    9    2  852    3    1    6    2]
 [   7    1    2    0    2    3  953    0    6    0]
 [   1    3    4    1    2    0    0 1035    1    6]
 [   4    2    4    7    0    3    0    3  992    3]
```

[1   1   1   6   7   3   0   9   5 947]]

Model 1 accuracy = 0.977

```
#model 2
#2 hidden layers
#sigmoid activation
#sigmoid output
model2 = Sequential()
model2.add(Dense(128, activation='sigmoid', input_shape=(784,)))
model2.add(Dropout(0.2))
model2.add(Dense(64, activation='sigmoid'))
model2.add(Dense(num_classes, activation='sigmoid'))

model2.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

model2.fit(xtrain, ytrain,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1)

ypred = model2.predict_classes(xval)

conf, acc, rec, prec = func_confusion_matrix(yval,ypred)
print(conf)
print(acc)
print(rec)
print(prec)
```

[[944   0   1   3   1   2   3   1   5   0]

[0 1033   5   1   2   0   0   2   6   5]

[2   1 976   6   5   0   5   4   10   1]

[2   0   12 975   1   24   0   10   5   3]

[3   0   3   0 1007   0   6   3   1   19]

[2   3   2   14   3 838   4   2   5   4]

[4   2   2   1   2   5 954   0   4   0]

[2   6   2   3   1   0   0 1032   0   7]

[1   4   7   10   0   5   2   2 983   4]

[2   2   1   8   8   4   0   12   8 935]]

Model 2 accuracy = 0.9677

```python
#model 3
#3 hidden layers
#relu activation
#softmax output
model3 = Sequential()
model3.add(Dense(256, activation='relu', input_shape=(784,)))
model3.add(Dropout(0.2))
model3.add(Dense(128, activation='relu'))
model3.add(Dropout(0.2))
model3.add(Dense(64, activation='relu'))
model3.add(Dense(num_classes, activation='softmax'))

model3.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

model3.fit(xtrain, ytrain,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1)

ypred = model3.predict_classes(xval)

conf, acc, rec, prec = func_confusion_matrix(yval,ypred)
print(conf)
print(acc)
print(rec)
print(prec)
```

[[955 0 2 1 0 0 1 0 1 0]

[0 1041 3 0 2 0 1 1 3 3]

[2 1 988 8 1 0 2 2 5 1]

[0 0 7 1012 0 7 0 2 2 2]

[1 0 3 0 1016 0 2 1 1 18]

[1 1 1 9 1 857 2 0 3 2]

[4 1 1 0 1 3 960 0 4 0]

[0 4 0 3 2 0 0 1030 1 13]

[1 1 4 10 0 3 0 2 992 5]

[1 1 0 3 6 6 0 4 2 957]]

Model 3 accuracy = 0.9808

Model 3 has the best accuracy.

Question II:

Confusion Matrix for model 3 on the test data:

[[971 1 1 0 1 1 2 1 2 0]

```
[   0 1125    3    1    0    0    2    0    4    0]
[   0    0 1016    4    2    0    2    4    4    0]
[   0    0    4  995    0    2    0    4    3    2]
[   2    0    1    1  958    0    3    1    1   15]
[   2    0    0   13    1  869    3    0    2    2]
[   3    2    0    0    3    7  942    0    1    0]
[   2    6    9    6    1    0    0  988    3   13]
[   1    1    3    6    1    4    0    2  953    3]
[   3    2    0    8    4    1    0    1    1  989]]
```

Accuracy of model 3:

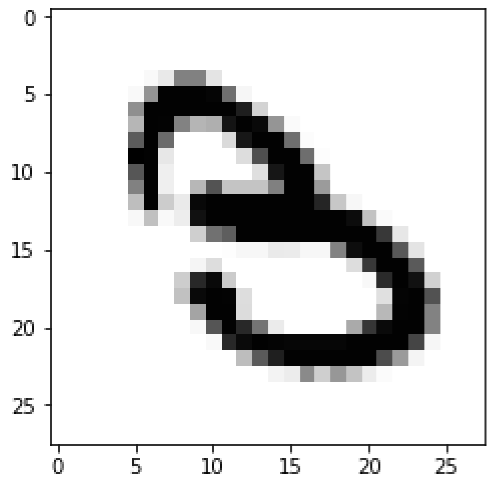
0.9806

Recall of model 3:

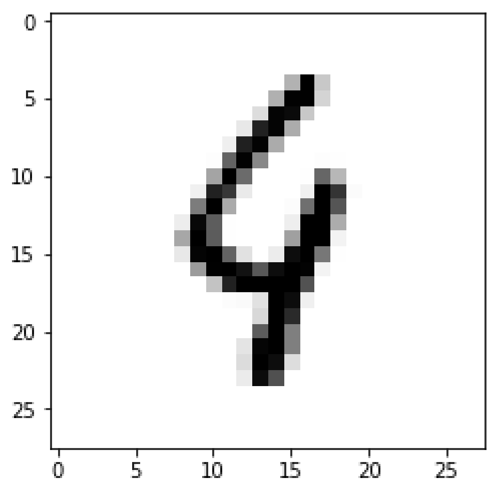
[0.99081633 0.99118943 0.98449612 0.98514851 0.97556008 0.97421525
 0.98329854 0.96108949 0.97843943 0.98017839]

Precision of model 3:

[0.98678862 0.98944591 0.97974928 0.9622824 0.98661174 0.98303167
 0.98742138 0.98701299 0.97843943 0.96582031]

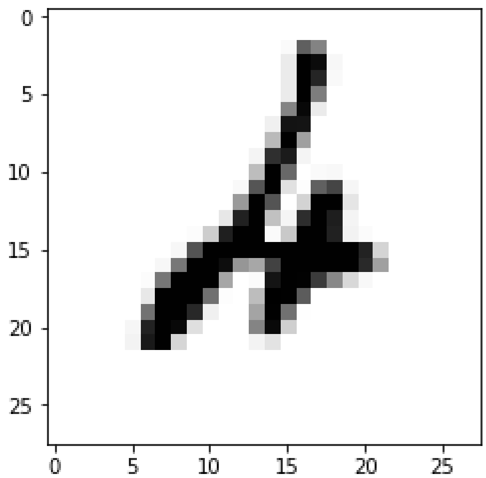



```
predicted 8
actual 3
```

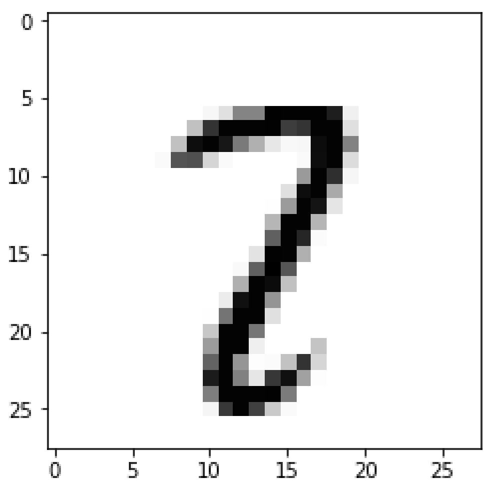



```
predicted 9
actual 4
```

If the lines were connected, the three would look a lot like an 8. Same with the top of the 4 and being similar to a 9.

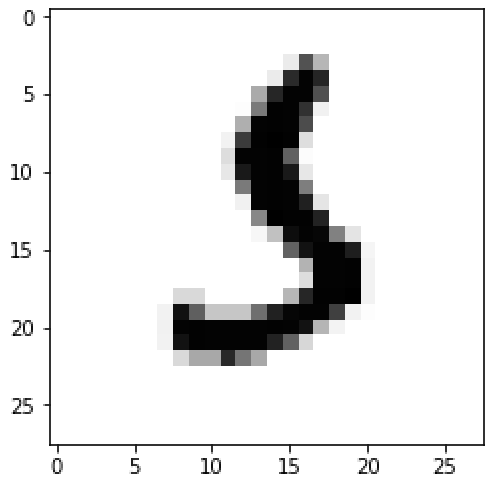


```
predicted 2
actual 4
```

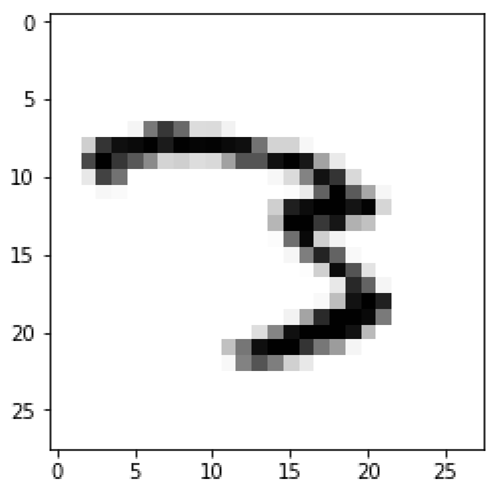


```
predicted 7
actual 2
```

The 4 is quite ambiguous so it would be hard to predict. This 2 looks similar to a 7.

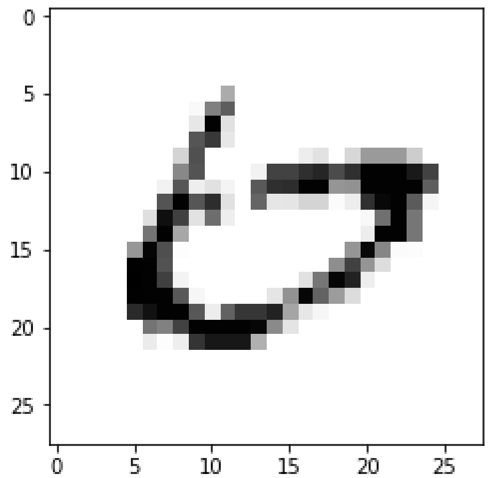



```
predicted 3
actual 5
```

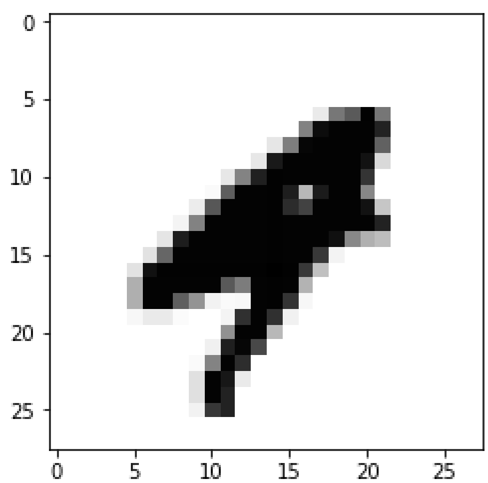



```
predicted 7
actual 3
```

This 5 looks like a 3 without the top part, and this 3 looks like a 7.

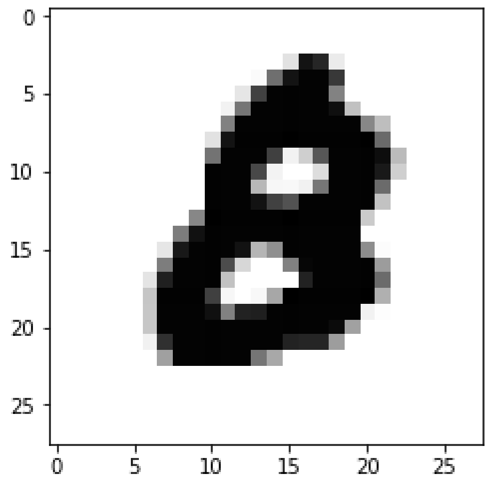



```
predicted 0
actual 6
```

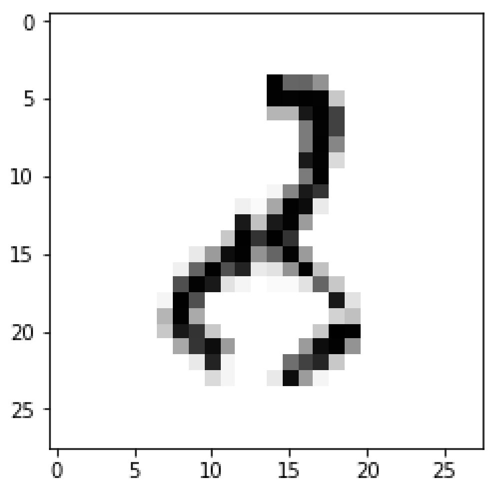



```
predicted 9
actual 4
```

The 6 is almost a 0, and the 4 could be a 9 or a 4.



predicted 2
actual 8

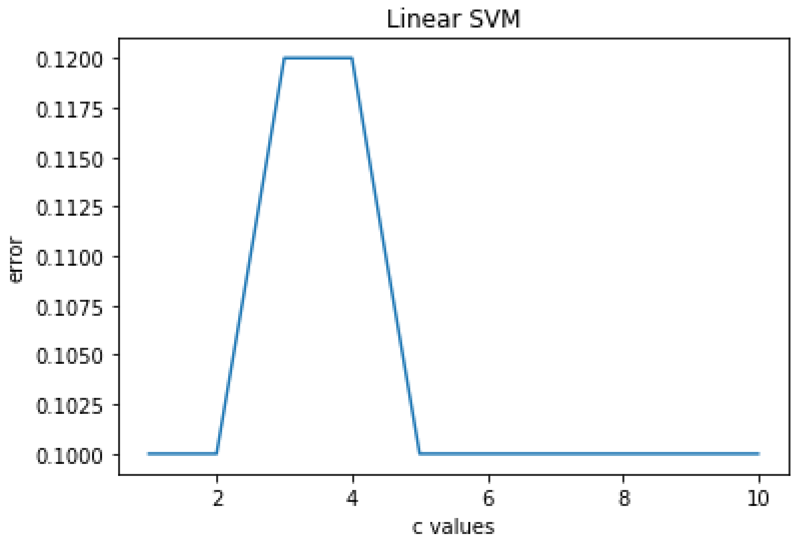


predicted 2
actual 8

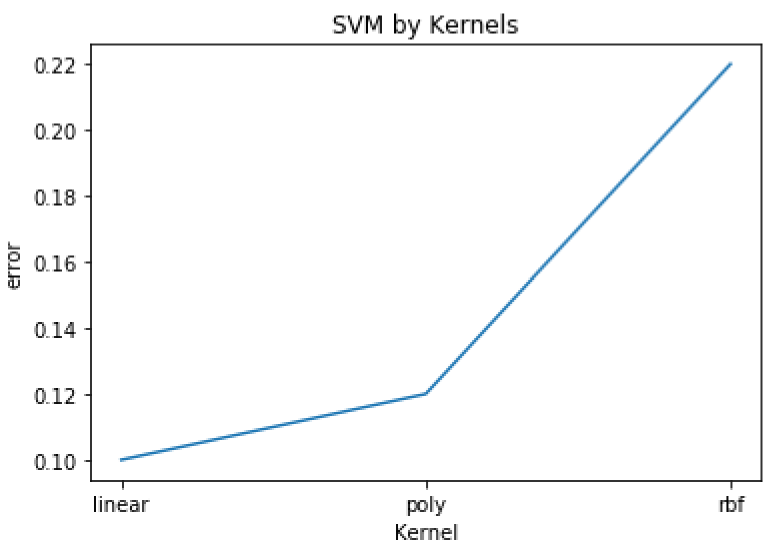
This 8 seems like a bad prediction, but the bottom 8 isn't even drawn well.

PART 2

A C value of 5 looks like it is the best for this data.



Linear performs better than poly and rbf.



Metrics for the test set:

```
Confusion Matrix:
[[52  0]
 [ 8 40]]
Average Accuracy: 0.92
Per-Class Precision: [0.86666667 1.        ]
Per-Class Recall: [1.         0.83333333]
```

Here are 5 incorrect predictions and 5 correct predictions. Correct predictions tend to have more extreme value features where the incorrect predictions are in the middle.

predicted 1.0

actual -1.0

data [1. 11.1 9.9 23.8 27.1 9.8]

predicted 1.0

actual -1.0

data [1. 12.3 11. 26.8 31.5 11.4]

predicted 1.0

actual -1.0

data [1.   9.2 7.8 19. 22.4 7.7]

predicted 1.0

actual -1.0

data [0.   9.1 6.9 16.7 18.6 7.4]

predicted 1.0

actual -1.0

data [1. 12.8 10.9 27.4 31.5 11.]

predicted -1.0

actual -1.0

data [1. 13.9 11.1 29.2 33.3 12.1]

predicted -1.0

actual -1.0

data [1. 19.8 14.2 43.2 49.7 18.6]

predicted -1.0

actual -1.0

data [1. 19.7 15.3 41.9 48.5 17.8]

predicted 1.0

actual 1.0

data [1. 14.7 12.5 30.1 34.7 12.5]

predicted 1.0

actual 1.0

data [0. 15.7 13.6 31. 34.8 13.8]