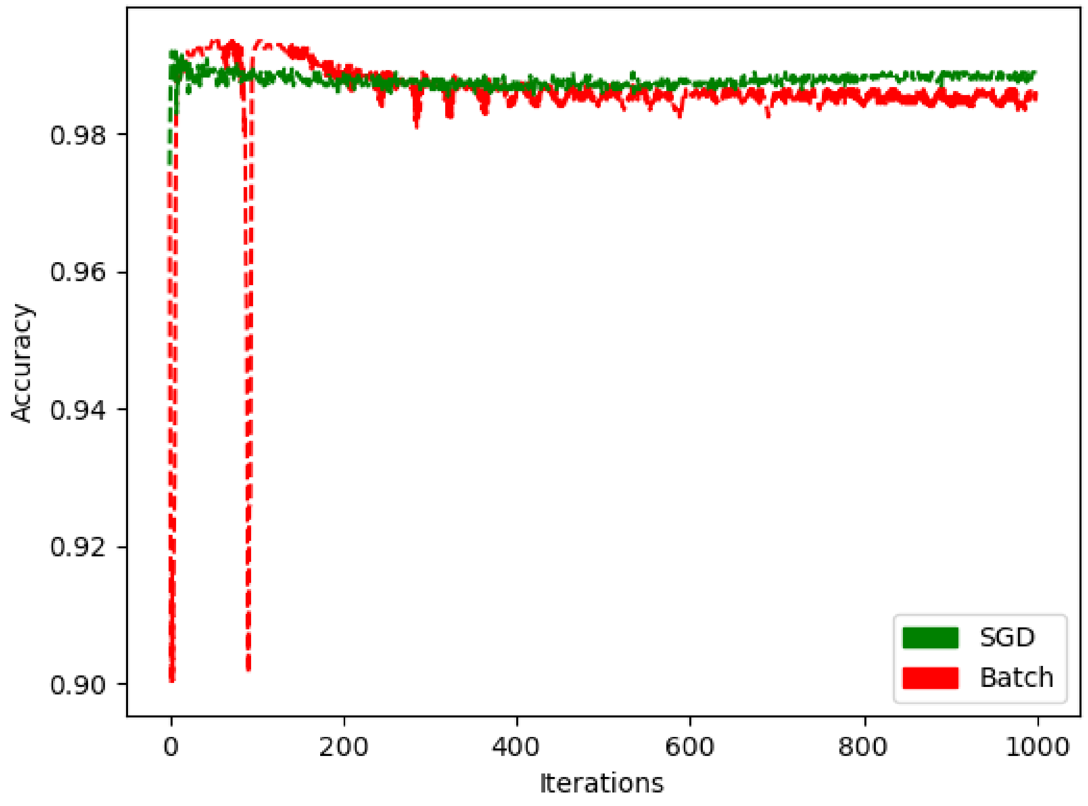
b,d) We implement an SVM using batch-gradient descent and SGD, and plot their performance relative to each other. Our code is as follows:

```
“import numpy as np
import pandas as pd
import random
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import csv
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

traindata= pd.read_csv(open("digits_training_data.csv"), header=None, sep = ' ').as_matrix()
trainlabels= pd.read_csv(open("digits_training_labels.csv"), header=None, sep = ' ').as_matrix()
testdata= pd.read_csv(open("digits_test_data.csv"), header=None, sep = ' ').as_matrix()
testlabels= pd.read_csv(open("digits_test_labels.csv"), header=None, sep = ' ').as_matrix()

trainlabels = trainlabels - 8 * np.ones((trainlabels.shape[0], 1))
testlabels = testlabels - 8 * np.ones((testlabels.shape[0], 1))

eta = 0.001
C = 3
N = traindata.shape[0]

iterBatch = 0
wBatch = np.zeros((traindata.shape[1],1))
ResidsBatch = np.zeros((N,1))
flagBatch = np.zeros((N,1))
accBatch = np.zeros(1000)
bBatch = 0.0
iterSGD = 0
wSGD = np.zeros((traindata.shape[1],1))
ResidSGD = 0.0
flagSGD = 0.0
accSGD = np.zeros(1000)
bSGD = 0.0

def learningRate(j, etaparam):
 return etaparam / (j * etaparam)

while iterBatch < 1000:
 iterBatch += 1
 wBatchOld = wBatch
 bBatchOld = bBatch
 A = learningRate(iterBatch, eta)
 for i in range(N):
  ResidsBatch[i] = trainlabels[i] * (traindata[i,:] @ wBatch + bBatch)
  if (ResidsBatch[i] < 1.0):
   flagBatch[i] = 1.0
  if (ResidsBatch[i] >= 1.0):
   flagBatch[i] = 0.0
 wGrad = wBatch - C * (np.transpose(traindata) @ np.multiply(flagBatch, trainlabels))
 bGrad = - C * (np.transpose(flagBatch) @ trainlabels)
 wBatch = wBatchOld - (A * wGrad)
 bBatch = bBatchOld - (A * bGrad)
 guessBatch = np.transpose(np.sign(np.transpose(testdata @ wBatch) + (bBatch * np.ones((testdata @
wBatch).shape[0]))))
 accB = 1 - np.count_nonzero((guessBatch - testlabels) / 2) / N
 accBatch[iterBatch - 1] = accB

while iterSGD < 1000:
 iterSGD += 1
 A = learningRate(iterSGD, eta)
 for i in random.sample(range(N), len(range(N))):
  wSGDOld = wSGD
  bSGDOld = bSGD
  sample = traindata[i,:].reshape(1,traindata.shape[1])
  ResidSGD = trainlabels[i][0] * ((sample @ wSGDOld)[0][0] + bSGDOld)
  if (ResidSGD < 1.0):
   flagSGD = 1.0
  if (ResidSGD >= 1.0):
   flagSGD = 0.0
  wGrad = wSGD/(2*N) - (C * trainlabels[i][0] * np.transpose(sample) * flagSGD)
  bGrad = -1*C * trainlabels[i][0] * flagSGD
  wSGD = wSGDOld - (A * wGrad)
  bSGD = bSGDOld - (A * bGrad)
 guessSGD = np.transpose(np.sign(np.transpose(testdata @ wSGD) + (bSGD * np.ones((testdata @
wSGD).shape[0]))))
 accS = 1 - np.count_nonzero((guessSGD - testlabels) / 2) / N
 accSGD[iterSGD - 1] = accS

k = range(1000)
plt.plot(k, accBatch, 'r--', k, accSGD, 'g--')
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
red_patch = mpatches.Patch(color='r', label='Batch')
green_patch = mpatches.Patch(color='g', label='SGD')
plt.legend(handles=[green_patch, red_patch])
plt.show()”
```



e) We note that, in general, SGD appears to converge more quickly than batch gradient descent. Though batch gradient descent performs better on short time intervals on this dataset, SGD performs consistently better after a longer period of time, indicating that it converges quicker and to a better value than batch gradient descent.

g) We kernelize the SVM. Our code and output are:

Code:

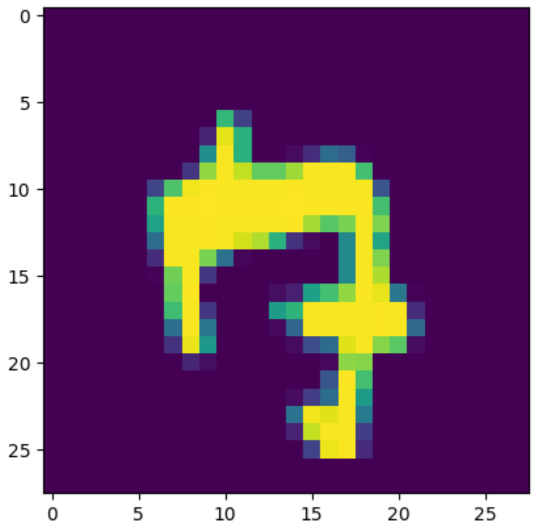
```
“from PIL import Image
import numpy as np
import pandas as pd
import random
from sklearn import svm
from sklearn import metrics
import csv
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

traindata= pd.read_csv(open("digits_training_data.csv"), header=None, sep = ' ').as_matrix()
trainlabels= pd.read_csv(open("digits_training_labels.csv"), header=None, sep = ' ').as_matrix()
testdata= pd.read_csv(open("digits_test_data.csv"), header=None, sep = ' ').as_matrix()
testlabels= pd.read_csv(open("digits_test_labels.csv"), header=None, sep = ' ').as_matrix()

trainlabels = trainlabels - 8 * np.ones((trainlabels.shape[0], 1))
testlabels = testlabels - 8 * np.ones((testlabels.shape[0], 1))

param_grid = dict(C=np.linspace(0.1, 2, 20))

print("Soft-Margin SVM:")
SoftClf = svm.SVC(C=3.0,kernel='rbf')
SoftClf.fit(traindata,trainlabels.ravel())
SoftTrainGuess=SoftClf.predict(traindata)
errstrain = metrics.accuracy_score(trainlabels,SoftTrainGuess)
print("The training accuracy is %f"
 % (errstrain))
SoftTestGuess=SoftClf.predict(testdata)
errstest = metrics.accuracy_score(testlabels,SoftTestGuess)
print("The test accuracy is %f"
 % (errstest))

for i in range(testlabels.shape[0]):
 if (SoftTestGuess[i] != testlabels[i]):
  plt.imshow(np.transpose(np.reshape(testdata[i,:],(28,28))), interpolation='nearest')
  print(SoftTestGuess[i])
  plt.show()”
```

Output:

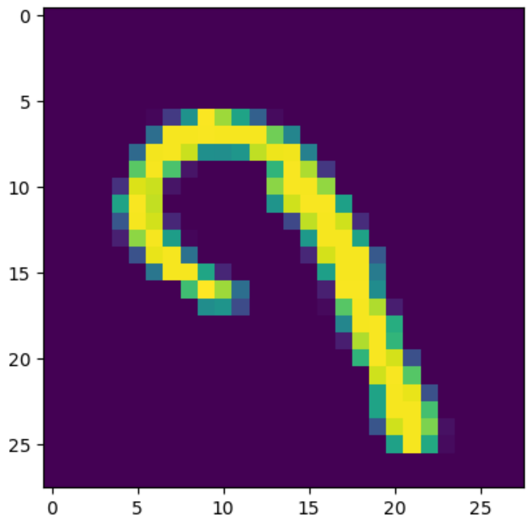
```
“Soft-Margin SVM:
The training accuracy is 0.963000
The test accuracy is 0.970000”
```

We report some misclassified images, and the values that the algorithm guessed for them:

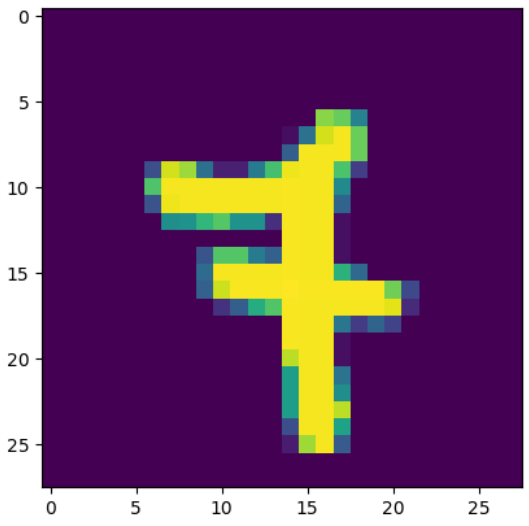
Guess: Nine



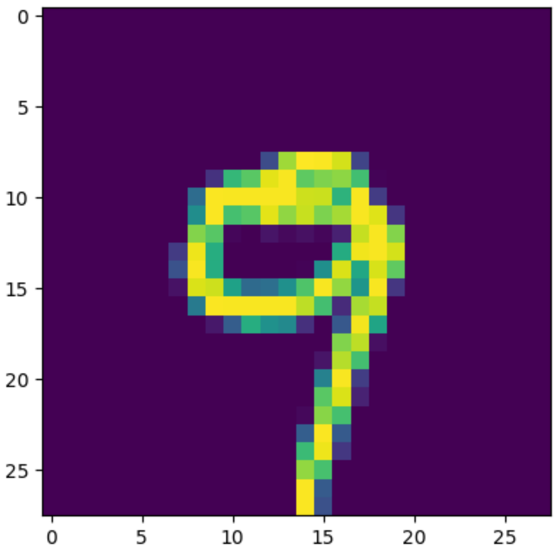
Guess: Seven



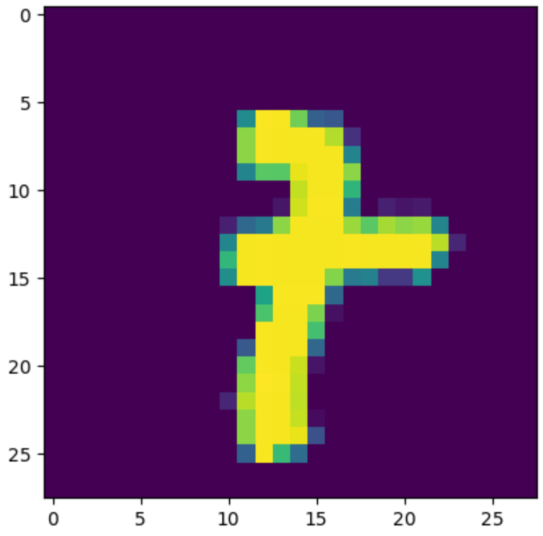
Guess: Nine



Guess: Seven



Guess: Nine


0
5
10
15
20
25
0
5
10
15
20
25