Use PyTorch (by switching to a different kernel) to build a simple fully-connected artificial neural network for the beans classification based on the chosen features provided in the data. Generate a confusion matrix for the test data set to demonstrate the accuracy of the model. Based on your model, classify the beans provided in the unlabeled beans-unknown.csv data set. Indicate which classification has been assigned to each of the unlabeled beans. How do the results with the artificial neural network compare to the support vector machine model?

In [62]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
from sklearn.metrics import confusion_matrix
from sklearn import svm
```

In [36]:

```
## Loading the dataset
bean = pd.read_csv('/public/bmort/python/beans.csv')
bean = bean[['Area', 'AspectRatio','Extent','Solidity', 'roundness','ShapeFactor4','Cla
ss']]
bean.head()
```

Out[36]:

	Area	AspectRatio	Extent	Solidity	roundness	ShapeFactor4	Class
0	28395	1.197191	0.763923	0.988856	0.958027	0.998724	SEKER
1	28734	1.097356	0.783968	0.984986	0.887034	0.998430	SEKER
2	29380	1.209713	0.778113	0.989559	0.947849	0.999066	SEKER
3	30008	1.153638	0.782681	0.976696	0.903936	0.994199	SEKER
4	30140	1.060798	0.773098	0.990893	0.984877	0.999166	SEKER

In [37]:

```
## Using the MinMaxScaler model to tranform the the selected columns
# create a scaler object
features_names = bean[['Area', 'AspectRatio', 'Extent', 'Solidity','roundness','ShapeFa
ctor4']]
scaler = MinMaxScaler()

# fit and transform the data
scaled = pd.DataFrame(scaler.fit_transform(features_names),
                      columns=features_names.columns)

scaled.head()
```

Out[37]:

	Area	AspectRatio	Extent	Solidity	roundness	ShapeFactor4
0	0.034053	0.122612	0.671024	0.922824	0.934823	0.980620
1	0.035500	0.051577	0.735504	0.871514	0.793138	0.974979
2	0.038259	0.131521	0.716671	0.932141	0.914511	0.987196
3	0.040940	0.091623	0.731365	0.761614	0.826871	0.893675
4	0.041504	0.025565	0.700538	0.949832	0.988408	0.989116

In [38]:

```
## Replacing the transformed columns into the main data
bean[['Area', 'AspectRatio', 'Extent', 'Solidity','roundness','ShapeFactor4']] = scaled
[['Area', 'AspectRatio', 'Extent',
                                                                    'Solidi
ty','roundness','ShapeFactor4']]
bean.head()
```

Out[38]:

	Area	AspectRatio	Extent	Solidity	roundness	ShapeFactor4	Class
0	0.034053	0.122612	0.671024	0.922824	0.934823	0.980620	SEKER
1	0.035500	0.051577	0.735504	0.871514	0.793138	0.974979	SEKER
2	0.038259	0.131521	0.716671	0.932141	0.914511	0.987196	SEKER
3	0.040940	0.091623	0.731365	0.761614	0.826871	0.893675	SEKER
4	0.041504	0.025565	0.700538	0.949832	0.988408	0.989116	SEKER

In [39]:

```
## Replacing the class columns with labels
l_eco = LabelEncoder()
l_eco.fit(bean['Class'])
bean['Class'] = l_eco.transform(bean['Class'])
```

In [40]:

```
## Splitting the data into the predictors and target varible
X = bean[['Area', 'AspectRatio','Extent','Solidity','roundness','ShapeFactor4']].to_num
py()
```

In [41]:

```
X
```

Out[41]:

```
array([[0.03405267, 0.12261211, 0.67102371, 0.92282385, 0.93482256,
        0.98061988],
       [0.03550018, 0.05157739, 0.73550396, 0.87151366, 0.79313798,
        0.97497943],
       [0.03825855, 0.13152124, 0.71667069, 0.9321406 , 0.9145106 ,
        0.98719586],
       ...,
       [0.09273856, 0.31855826, 0.56168866, 0.93664773, 0.85578518,
        0.9430251 ],
       [0.09277272, 0.33047232, 0.48274074, 0.90899135, 0.83479471,
        0.91334232],
       [0.09282396, 0.42333667, 0.75156921, 0.93332197, 0.7958257 ,
        0.9701623 ]])
```

In [42]:

```
y = bean['Class'].to_numpy()
```

In [43]:

```
## Train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2)
```

In [44]:

```
# input layer nodes = 6 = number of features
# hidden layer nodes = 3
# output layer nodes = 6 = number of categories
iln = 6
hln = 10
oln = 7
l_r = 0.01
num_epoch = 100000
```

In [45]:

```
## fully connected
class NNetwork(nn.Module):
    def __init__(self):
        super(NNetwork, self).__init__()
        self.fc1 = nn.Linear(iln, hln)
        self.out = nn.Linear(hln, oln)
    def forward(self, x):
        x = self.fc1(x)
        x = nn.functional.relu(x)
        x = self.out(x)
        out = nn.functional.softmax(x, dim = 1)
        return out
```

In [46]:

```
## Setting up the classifer
classifier = NNetwork()
```

In [47]:

```
## Defining loss function and optimizers
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(classifier.parameters(), lr=l_r )
```

In [48]:

```
X_torch = torch.Tensor(X_train).float()
y_torch = torch.Tensor(y_train).long()
# X_torch
# y_torch.unique()
```

In [49]:

```
## Training our data
for epoch in range(num_epoch):
    optimizer.zero_grad()
    output = classifier(X_torch)
    loss = criterion(output, y_torch)
    loss.backward()
    optimizer.step()
    if epoch % 1000 == 0:
        print('epoch:', epoch, 'loss:', loss.item())
```

```
epoch: 0 loss: 1.9462968111038208
epoch: 1000 loss: 1.925895094871521
epoch: 2000 loss: 1.8966991901397705
epoch: 3000 loss: 1.8864113092422485
epoch: 4000 loss: 1.883758544921875
epoch: 5000 loss: 1.881467580795288
epoch: 6000 loss: 1.8793209791183472
epoch: 7000 loss: 1.8774968385696411
epoch: 8000 loss: 1.8759421110153198
epoch: 9000 loss: 1.8744826316833496
epoch: 10000 loss: 1.8729615211486816
epoch: 11000 loss: 1.8712364435195923
epoch: 12000 loss: 1.8691270351409912
epoch: 13000 loss: 1.866313099861145
epoch: 14000 loss: 1.8620822429656982
epoch: 15000 loss: 1.8547999858856201
epoch: 16000 loss: 1.84213387966156
epoch: 17000 loss: 1.8248512744903564
epoch: 18000 loss: 1.807938814163208
epoch: 19000 loss: 1.7948945760726929
epoch: 20000 loss: 1.7857009172439575
epoch: 21000 loss: 1.7790424823760986
epoch: 22000 loss: 1.7736531496047974
epoch: 23000 loss: 1.7685465812683105
epoch: 24000 loss: 1.7633264064788818
epoch: 25000 loss: 1.7584114074707031
epoch: 26000 loss: 1.7540169954299927
epoch: 27000 loss: 1.7500808238983154
epoch: 28000 loss: 1.7465323209762573
epoch: 29000 loss: 1.7432997226715088
epoch: 30000 loss: 1.7403144836425781
epoch: 31000 loss: 1.7375150918960571
epoch: 32000 loss: 1.734842300415039
epoch: 33000 loss: 1.7322324514389038
epoch: 34000 loss: 1.7296018600463867
epoch: 35000 loss: 1.726828932762146
epoch: 36000 loss: 1.7237623929977417
epoch: 37000 loss: 1.7203843593597412
epoch: 38000 loss: 1.7170002460479736
epoch: 39000 loss: 1.7137846946716309
epoch: 40000 loss: 1.7106521129608154
epoch: 41000 loss: 1.70755934715271
epoch: 42000 loss: 1.70449960231781
epoch: 43000 loss: 1.7014691829681396
epoch: 44000 loss: 1.6984648704528809
epoch: 45000 loss: 1.695483684539795
epoch: 46000 loss: 1.692525029182434
epoch: 47000 loss: 1.6895869970321655
epoch: 48000 loss: 1.6866706609725952
epoch: 49000 loss: 1.6837763786315918
epoch: 50000 loss: 1.6809048652648926
epoch: 51000 loss: 1.6780563592910767
epoch: 52000 loss: 1.6752299070358276
epoch: 53000 loss: 1.6724274158477783
epoch: 54000 loss: 1.6696488857269287
epoch: 55000 loss: 1.6668895483016968
epoch: 56000 loss: 1.6641532182693481
epoch: 57000 loss: 1.6614363193511963
epoch: 58000 loss: 1.6587368249893188
epoch: 59000 loss: 1.656058430671692
epoch: 60000 loss: 1.6534103155136108
```

```
epoch: 61000 loss: 1.6507971286773682
epoch: 62000 loss: 1.6482312679290771
epoch: 63000 loss: 1.6457221508026123
epoch: 64000 loss: 1.6432819366455078
epoch: 65000 loss: 1.6409032344818115
epoch: 66000 loss: 1.6385836601257324
epoch: 67000 loss: 1.6363210678100586
epoch: 68000 loss: 1.6341071128845215
epoch: 69000 loss: 1.631937861442566
epoch: 70000 loss: 1.629806637763977
epoch: 71000 loss: 1.6277104616165161
epoch: 72000 loss: 1.6256434917449951
epoch: 73000 loss: 1.6236003637313843
epoch: 74000 loss: 1.621577262878418
epoch: 75000 loss: 1.619568943977356
epoch: 76000 loss: 1.6175792217254639
epoch: 77000 loss: 1.615607500076294
epoch: 78000 loss: 1.6136534214019775
epoch: 79000 loss: 1.6117160320281982
epoch: 80000 loss: 1.609796404838562
epoch: 81000 loss: 1.6078932285308838
epoch: 82000 loss: 1.606009840965271
epoch: 83000 loss: 1.6041470766067505
epoch: 84000 loss: 1.6023049354553223
epoch: 85000 loss: 1.6004807949066162
epoch: 86000 loss: 1.5986775159835815
epoch: 87000 loss: 1.5968947410583496
epoch: 88000 loss: 1.5951292514801025
epoch: 89000 loss: 1.5933809280395508
epoch: 90000 loss: 1.5916491746902466
epoch: 91000 loss: 1.589935064315796
epoch: 92000 loss: 1.5882388353347778
epoch: 93000 loss: 1.5865586996078491
epoch: 94000 loss: 1.584896206855774
epoch: 95000 loss: 1.583250641822815
epoch: 96000 loss: 1.581620454788208
epoch: 97000 loss: 1.580006718635559
epoch: 98000 loss: 1.5784075260162354
epoch: 99000 loss: 1.576823115348816
```

In [50]:

```
X_ttest = torch.Tensor(X_test).float()
y_ttest = torch.Tensor(y_test).long()
```

In [51]:

```
f_output = classifier(X_ttest)
```

In [52]:

```
bean_new = pd.read_csv('/public/bmort/python/beans-unknown.csv')
```

In [53]:

```
(values, pred) = torch.max(f_output.data, dim = 1 )
```

In [54]:

```
pred
```

Out[54]:

```
tensor([6, 2, 3,  ..., 3, 3, 4])
```

In [55]:

```
print(f'The Accuracy of the model is {(torch.sum(y_ttest==pred)/len(y_ttest))*100}%')
```

```
The Accuracy of the model is 62.68932342529297%
```

In [56]:

```
confusion_matrix(y_ttest, pred)
```

Out[56]:

```
array([[  0,   0, 180,   1,  11,   0,  89],
       [  0,   0, 117,   0,   0,   0,   0],
       [  0,   0, 313,   2,   7,   0,  14],
       [  0,   0,   0, 609,   2,   0,  65],
       [  0,   0,  19,   1, 374,   0,   3],
       [  0,   0,   0, 375,   0,   0,  14],
       [  0,   0,   6,  99,   5,   0, 401]])
```

In [71]:

```
new_data = pd.read_csv('/public/bmort/python/beans-unknown.csv')[['Area', 'AspectRati
o', 'Extent', 'Solidity','roundness','ShapeFactor4']]
```

In [72]:

```
new_data
```

Out[72]:

	Area	AspectRatio	Extent	Solidity	roundness	ShapeFactor4
0	37500	1.586948	0.703406	0.988299	0.888690	0.995836
1	37500	1.549351	0.786229	0.992142	0.920295	0.998631
2	37511	1.490660	0.717365	0.990573	0.913474	0.998379
3	37513	1.518721	0.780545	0.987678	0.909270	0.998076
4	37514	1.521158	0.793309	0.989293	0.894773	0.997545

In [90]:

```
X_newd = torch.tensor(new_data.to_numpy()).float()
```

In [92]:

```
## Predicting the new dataset
classifier(X_newd)
```

Out[92]:

```
tensor([[0., 0., 1., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0.]], grad_fn=<SoftmaxBackward>)
```

Fitting the SVM model

In [77]:

```
# Creating an SVM classifier with a linear kernel
svm_clf = svm.SVC(kernel='linear')

##training the model
svm_clf.fit(X_train,y_train)
```

Out[77]:

```
SVC(kernel='linear')
```

In [78]:

```
svm_clf.predict(X_test)
```

Out[78]:

```
array([6, 1, 3, ..., 3, 5, 4])
```

In [83]:

```
svm_clf.score(X,y)
```

Out[83]:

```
0.9133968816965935
```

In [85]:

```
print(f'The accuracy for the SVM model is {round(svm_clf.score(X,y),4)*100}%')
```

```
The accuracy for the SVM model is 91.34%
```

In [80]:

```
## Predicting the new model
X_new = new_data.to_numpy()
```

In [86]:

```
## Prediciting the new dataset
svm_clf.predict(X_new)
```

Out[86]:

```
array([1, 1, 1, 1, 1])
```

Comparing the two models, we can see that the SVM model has a higher accuracy than the neural network model and therefore can guarantee us accurate classifications of objects into their actual groups.