Solution 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import sqrtm


# Define the adjacency matrix for the graph
adjacency_matrix = np.matrix(np.array([
     [0, 1, 0, 0, 1, 1],
     [1, 0, 1, 1, 0, 0],
     [0, 1, 0, 1, 0, 0],
     [0, 1, 1, 0, 0, 0],
     [1, 0, 0, 0, 0, 0],
     [1, 0, 0, 0, 0, 0]
]))


# Calculate the degree matrix
degree_matrix = np.matrix(np.diag(np.sum(np.array(adjacency_matrix), axis=0)))
degree_matrix
```

```
matrix([[3, 0, 0, 0, 0, 0],
        [0, 3, 0, 0, 0, 0],
        [0, 0, 2, 0, 0, 0],
        [0, 0, 0, 2, 0, 0],
        [0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 0, 1]])
```

1.1: *Consider a binary classification problem. Write down the initial label vector P0 for this graph where v6 has observed label 1 and v4 and v5 have observed label 2.*

```
# Set the initial labels for the graph
initial_labels = np.matrix(np.array([0, 0, 0, -1, -1, 1])).T
initial_labels
```

```
matrix([[ 0],
        [ 0],
        [ 0],
        [-1],
        [-1],
        [ 1]])
```

```
# Normalize the similarity matrix
normalized_similarity_matrix = sqrtm(np.linalg.matrix_power(degree_matrix, -1)) @ adjacency_matrix @ sqrtm(np.linalg.matrix_po


# Implement the label spreading algorithm
def label_spread(similarity_matrix, labels, decay=0.8, max_iter=20):
    m = 0
    for i in range(max_iter):
        aS = decay * similarity_matrix
        m += np.linalg.matrix_power(aS, i) @ labels
    aS = decay * similarity_matrix
    final_labels = (1 - decay) * m + np.linalg.matrix_power(aS, max_iter) @ labels
    return final_labels


# Function to get labels for nodes
def get_labels_vector(labels):
    labels_vector = ['' for _ in range(labels.shape[0])]
    for i in range(labels.shape[0]):
        if labels[i] < 0:
            labels_vector[i] = "label 2"
        elif labels[i] > 0:
            labels_vector[i] = "label 1"
        else:
            labels_vector[i] = "No label"
    return labels_vector

    # Function to print labels
def print_labels(labels_vector):
    print("v1:", str(labels_vector[0]), "\nv2:", labels_vector[1], "\nv3:", labels_vector[2])
```

1.2: *Perform 1 iteration of the label spreading algorithm with the decay parameter α = 0.8 and determine the node labels for the unlabeled nodes v1, v2, and v3, i.e., compute P1 and provide the labels l1, l2, and l3 after 1 iteration.*

```
#Perform 1 iteration of the label spreading algorithm
labels_after_1_iteration = label_spread(normalized_similarity_matrix, initial_labels, 0.8, 1)
labels_vector_1 = get_labels_vector(labels_after_1_iteration)
print_labels(labels_vector_1)
```

```
v1: No label
v2: label 2
v3: label 2
```

1.3: *Perform 2 iterations of the label spreading algorithm with the decay parameter α = 0.8 and determine the node labels for the unlabeled nodes v1, v2, and v3, i.e., compute P2 and provide the labels l1, l2, and l3 after 2 iterations.*

```
#Perform 2 iterations of the label spreading algorithm
labels_after_2_iterations = label_spread(normalized_similarity_matrix, initial_labels, 0.8, 2)
labels_vector_2 = get_labels_vector(labels_after_2_iterations)
print_labels(labels_vector_2)
```

```
v1: label 2
v2: label 2
v3: label 2
```

1.4: *Perform infinite iterations of the label spreading algorithm with the decay parameter α = 0.8 and determine the node labels for the unlabeled nodes v1, v2, and v3, i.e., compute P∞ and provide the labels l1, l2, and l3 after infinite iterations.*

```
#Perform infinite iterations of the label spreading algorithm
labels_after_infinite_iterations = label_spread(normalized_similarity_matrix, initial_labels, 0.8, 10000)
labels_vector_infinite = get_labels_vector(labels_after_infinite_iterations)
print_labels(labels_vector_infinite)
```

```
v1: label 2
v2: label 2
v3: label 2
```

We will verify convergence through the utilization of the following formula:

$F_* = (I - \alpha S)^{-1} * Y$

```
# Confirm convergence
convergence_check = (1 - 0.8) * np.linalg.matrix_power((np.identity(normalized_similarity_matrix.shape[0]) - 0.8 * normalized_
convergence_check
```

```
matrix([[-0.0972985 ],
        [-0.20919178],
        [-0.20910767],
        [-0.35196482],
        [-0.24494025],
        [ 0.15505975]])
```

we can confirm the convergence from the above result.

1.5: *Determine the node labels for the unlabeled nodes v1, v2, and v3 via the energy minimization algorithm.*

lets first calculate the laplacian matrix for the Similarity matrix:

```
laplacian_matrix = degree_matrix - adjacency_matrix
laplacian_matrix
```

```
matrix([[ 3, -1,  0,  0, -1, -1],
        [-1,  3, -1, -1,  0,  0],
        [ 0, -1,  2, -1,  0,  0],
        [ 0, -1, -1,  2,  0,  0],
        [-1,  0,  0,  0,  1,  0],
        [-1,  0,  0,  0,  0,  1]])
```

```
# One-hot vector encoding of the labeled nodes
one_hot_labels = np.matrix([
    [0, 1],
    [0, 1],
    [1, 0]
])
```

We will use below formula to calculate the Fu

$$F_u = -L_{uu}^{-1} L_{uu} Y_l$$

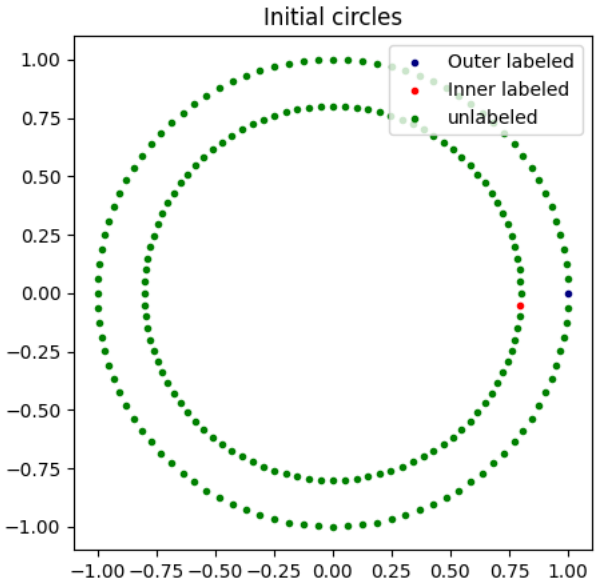
```
# Calculate Fu using the energy minimization algorithm
Fu = -1 * np.linalg.inv(laplacian_matrix[0:3, 0:3]) @ laplacian_matrix[3:6, 0:3] @ one_hot_labels
unlabeled_node_labels = np.argmax(Fu, axis=1)

# Print the results
unlabeled_node_labels
```

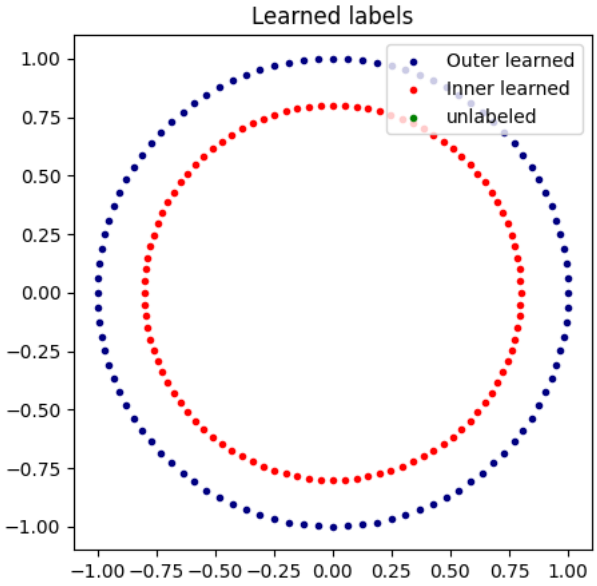
```
matrix([[1],
        [1],
        [1]])
```

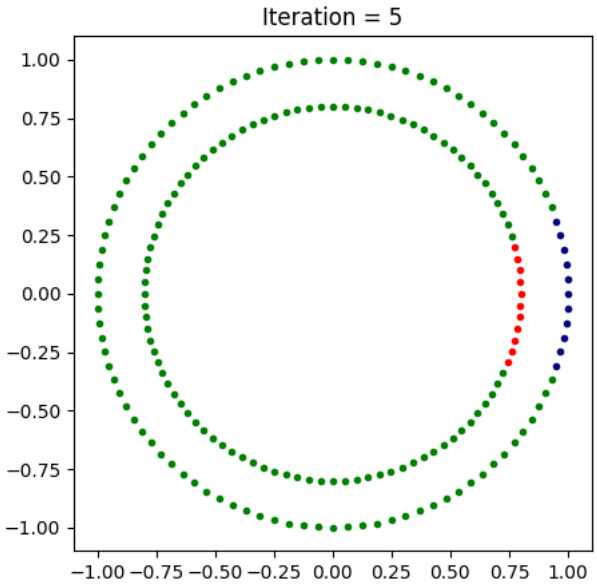
The analysis reveals that all data points are categorized under the second label. Furthermore, it is observed that the algorithm successfully identifies unlabeled points without altering the labeled nodes, in contrast to the label spreading algorithm, which alters the labels of the nodes.

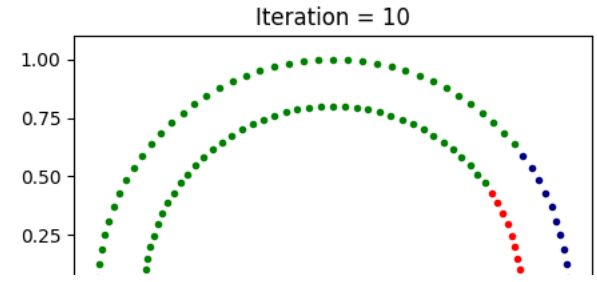
Solution 2:

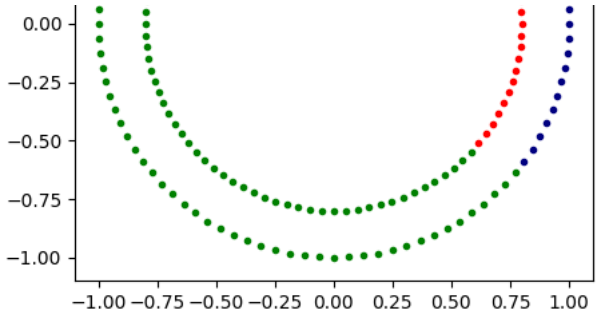
```
import numpy as np
from sklearn.datasets import make_circles
import matplotlib.pyplot as plt
from sklearn.neighbors import kneighbors_graph

# Generate two circles dataset
num_samples = 200
X, y = make_circles(n_samples=num_samples, shuffle=False)
class_outer, class_inner = 0, 1
labels = np.full(num_samples, -1.0)
labels[0] = class_outer
labels[-1] = class_inner

# Plot initial circles
def plot_scatter(data, target_labels, outer_class, inner_class, title="", outer_title="", inner_title="", legend=True):
    plt.figure(figsize=(5, 5))
    plt.scatter(data[target_labels == outer_class, 0], data[target_labels == outer_class, 1], color="navy", marker=".", label=
    plt.scatter(data[target_labels == inner_class, 0], data[target_labels == inner_class, 1], color="red", marker=".", label=i
    plt.scatter(data[target_labels == -1, 0], data[target_labels == -1, 1], color="green", marker=".", label="unlabeled")
    if legend:
        plt.legend(scatterpoints=1, shadow=False, loc="upper right")
    plt.title(title)
    plt.show()

plot_scatter(X, labels, class_outer, class_inner, title="Initial circles", outer_title="Outer labeled", inner_title="Inner lab

# Implement Label Spreading algorithm
def label_spreading(data, target_labels, alpha=0.8, tolerance=0.001, max_iterations=300):
    graph_matrix = kneighbors_graph(data, 3, mode='connectivity', include_self=True)
    classes = np.array([0, 1])
    Y1 = np.zeros((len(target_labels), len(classes)))
    for label in classes:
        Y1[target_labels == label, classes == label] = 1
    Y0 = np.copy(Y1) * (1 - alpha)
    Y_previous = np.zeros((data.shape[0], len(classes)))
    for _ in range(max_iterations):
        if np.abs(Y1 - Y_previous).sum() < tolerance:
            break
        Y_previous = Y1
        Y1 = graph_matrix @ Y1
        Y1 = alpha * Y1 + Y0
    out_labels = np.zeros(Y1.shape[0])
    for i in range(Y1.shape[0]):
        if Y1[i, 0] == 0 and Y1[i, 1] == 0:
            out_labels[i] = -1
        else:
            out_labels[i] = classes[np.argmax(Y1[i])]
    return out_labels

# Apply Label Spreading and plot results
output_labels = label_spreading(X, labels, max_iterations=100)
plot_scatter(X, output_labels, class_outer, class_inner, title="Learned labels", outer_title="Outer learned", inner_title="Inn

# Plot learning progress for iterations
for i in range(5, 51, 5):
    plot_scatter(X, label_spreading(X, labels, max_iterations=i), class_outer, class_inner, title="Iteration = " + str(i), leg
```

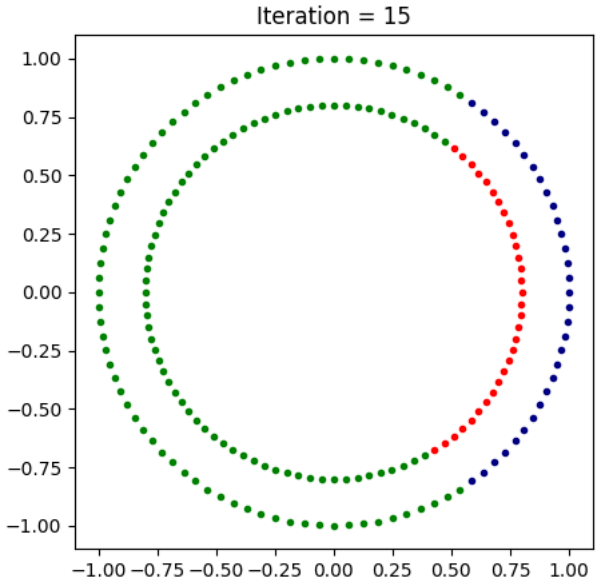

Initial circles
Outer labeled
Inner labeled
unlabeled
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

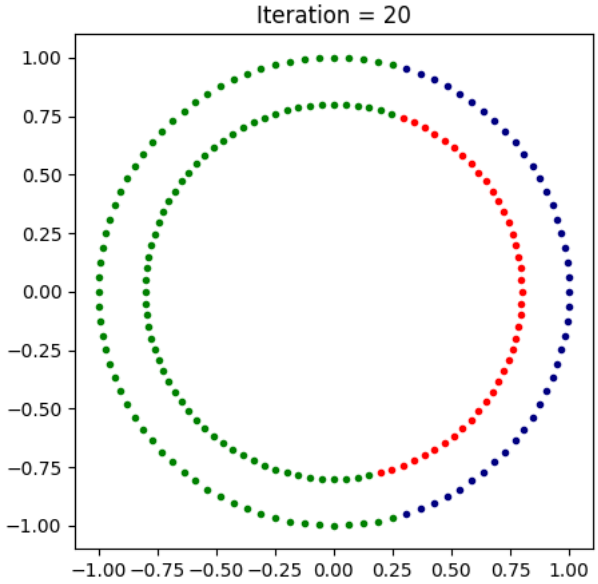

Learned labels
Outer learned
Inner learned
unlabeled
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

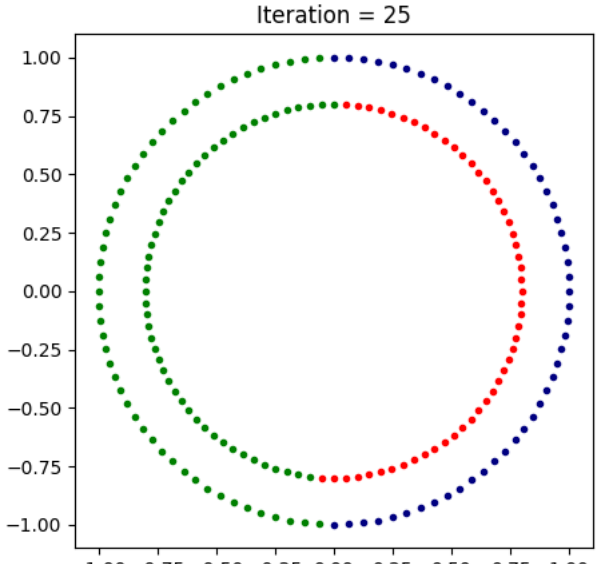

Iteration = 5
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

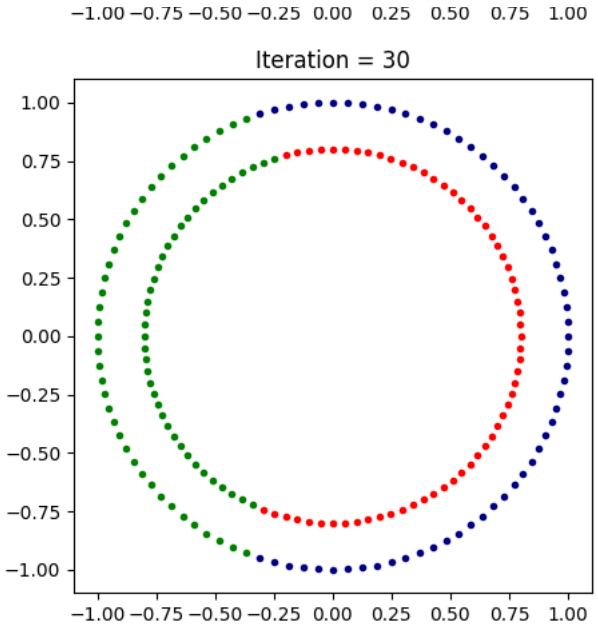

Iteration = 10
1.00
0.75
0.50
0.25

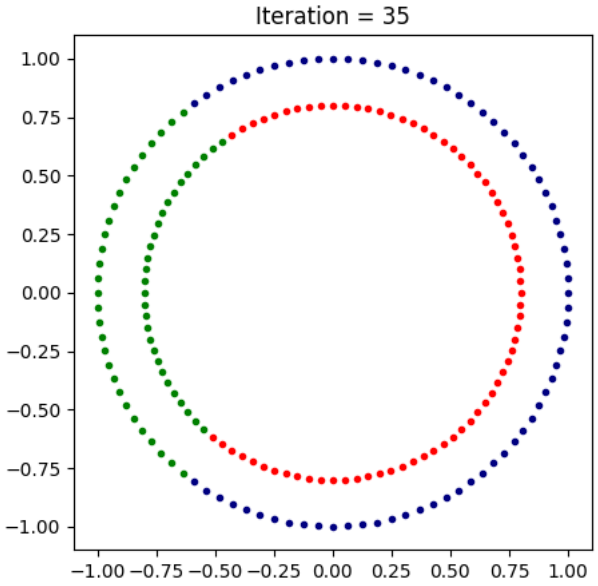

0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

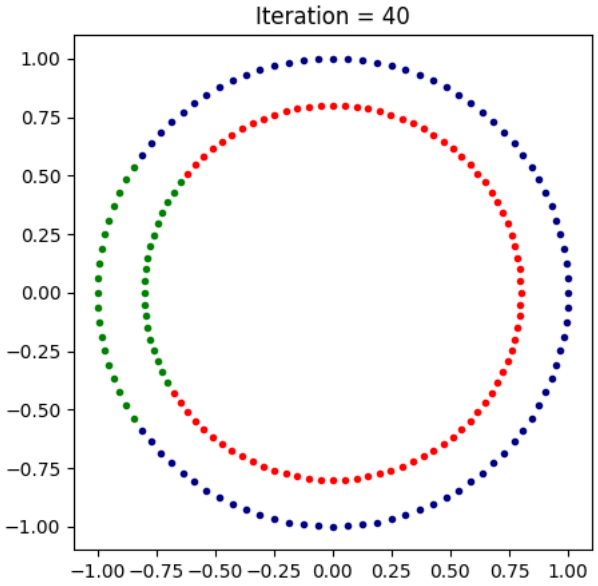

Iteration = 15
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

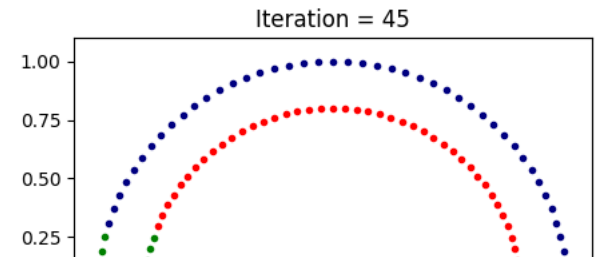

Iteration = 20
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

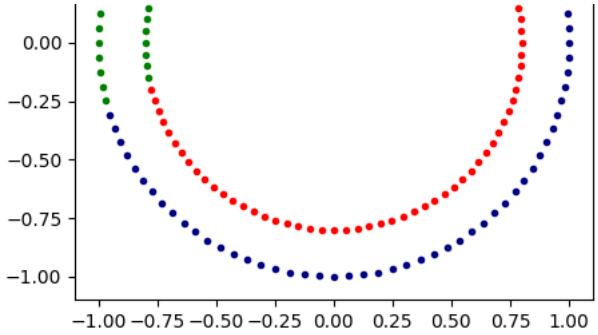

Iteration = 25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

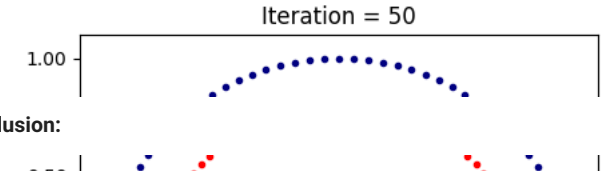

Iteration = 30
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00


Iteration = 35
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00


Iteration = 40
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00

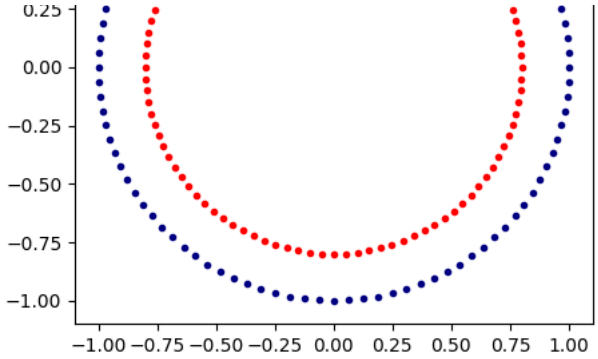

Iteration = 45
1.00 0.75 0.50 0.25





Conclusion:

The analysis demonstrates the accurate propagation of labels, revealing successful convergence of the model after 50 iterations.



Solution 3

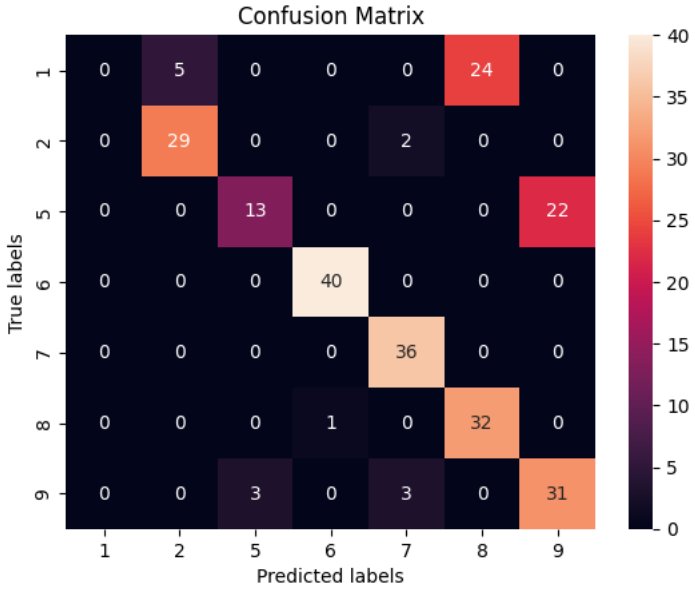
```
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.neighbors import kneighbors_graph
from scipy import stats
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns

# Generate data with 330 images of digits
digits = datasets.load_digits()
rng = np.random.RandomState(0)
indices = np.arange(len(digits.data))
rng.shuffle(indices)

X = digits.data[indices[:330]]
y = digits.target[indices[:330]]
images = digits.images[indices[:330]]

n_total_samples = len(y)
n_labeled_points = 10
unlabeled_data_set = np.arange(n_total_samples)[n_labeled_points:]

# Function for label propagation
def LabelPropagation(X, y, tolerence=0.001, max_iter=300):
    graph_matrix = kneighbors_graph(X, 7, mode='connectivity', include_self=True)
    classes = np.unique(y)
    classes = classes[classes != -1]
    y = np.asarray(y)
    unlabeled = y == -1

    Y1 = np.zeros((len(y), len(classes)))
    for label in classes:
        Y1[y == label, classes == label] = 1

    Y0 = np.copy(Y1)
    Y_prev = np.zeros((X.shape[0], len(classes)))
    unlabeled = unlabeled[:, np.newaxis]

    for n_iter_ in range(max_iter):
        if np.abs(Y1 - Y_prev).sum() < tolerence:
            break

        Y_prev = Y1
        Y1 = graph_matrix @ Y1
        normalizer = np.sum(Y1, axis=1)[:, np.newaxis]
        normalizer[normalizer == 0] = 1
        Y1 /= normalizer
        Y1 = np.where(unlabeled, Y1, Y0)

    F_u = classes[np.argmax(Y1, axis=1)]
    return F_u, classes, Y1


    # Display heat map of the Confusion matrix
def CM(cm, labels):
    ax = plt.subplot()
    sns.heatmap(cm, annot=True, fmt='g', ax=ax)
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix')
    ax.xaxis.set_ticklabels(labels)
    ax.yaxis.set_ticklabels(labels)
    plt.show()

    # Print model details
def PrintModel(true_labels, predicted_labels, labels, labeled_points):
    cm = confusion_matrix(true_labels, predicted_labels, labels=labels)
    a = accuracy_score(true_labels, predicted_labels)

    print("Label propagation model: %d labeled & %d unlabeled points (%d total)" % (labeled_points, n_total_samples - labeled_
    print("Accuracy: ", '{:.1%}'.format(a))
    print("Confusion matrix")
    print(cm)
    print("\n")
    CM(cm, labels)
    print("\n")
```

```
    # Plot certain digits of high confidence variables
def plotCertainNumbers(certain_index, Fu, y):
    f = plt.figure(figsize=(6, 5))
    plt.suptitle("")
    for index, image_index in enumerate(certain_index):
        image = images[image_index]
        sub = f.add_subplot(1, 5, index + 1)
        sub.imshow(image, cmap=plt.cm.gray_r)
        plt.xticks([])
        plt.yticks([])
        sub.set_title('predict: %i\ntrue: %i' % (Fu[image_index], y[image_index]))
    plt.show()
```

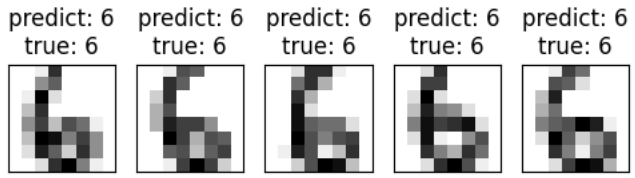
We execute a learning algorithm in which we select the next set of labeled predictors by choosing 5 predicted labels with high confidence.

```
# Learning algorithm with high confidence predicted labels
for i in range(5):
    y_train = np.copy(y)
    y_train[unlabeled_data_set] = -1

    Fu, labels, Y1 = LabelPropagation(X, y_train, tolerence=0.001, max_iter=300)

    predicted_labels = Fu[unlabeled_data_set]
    true_labels = y[unlabeled_data_set]

    PrintModel(true_labels, predicted_labels, labels, n_labeled_points)

    pred_entropies = stats.distributions.entropy(Y1.T)
    certain_index = np.argsort(pred_entropies)[::1]
    certain_index = certain_index[np.in1d(certain_index, unlabeled_data_set)][:5]

    plotCertainNumbers(certain_index, Fu, y)

    delete_indices = np.array([], dtype=int)
    unlabeled_data_set = np.setdiff1d(unlabeled_data_set, certain_index)
    n_labeled_points+= len(certain_index)
```

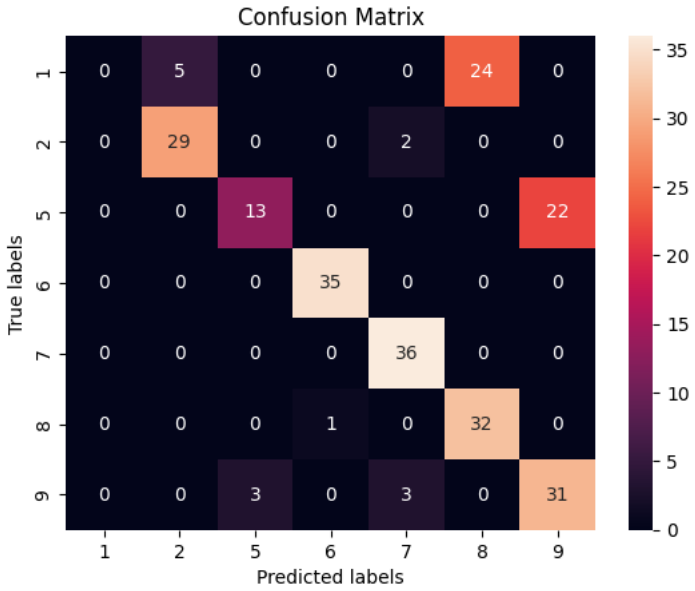
```
Label propagation model: 10 labeled & 320 unlabeled points (330 total)
Accuracy:  56.6%
Confusion matrix
[[ 0  5  0  0  0 24  0]
 [ 0 29  0  0  2  0  0]
 [ 0  0 13  0  0  0 22]
 [ 0  0  0 40  0  0  0]
 [ 0  0  0  0 36  0  0]
 [ 0  0  0  1  0 32  0]
 [ 0  0  3  0  3  0 31]]
```



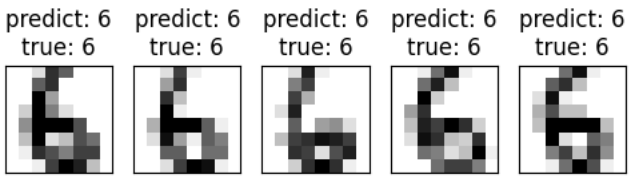

```
/usr/local/lib/python3.10/dist-packages/scipy/stats/_entropy.py:133: RuntimeWarning: invalid value encountered in divide
  pk = 1.0*pk / np.sum(pk, axis=axis, keepdims=True)
```



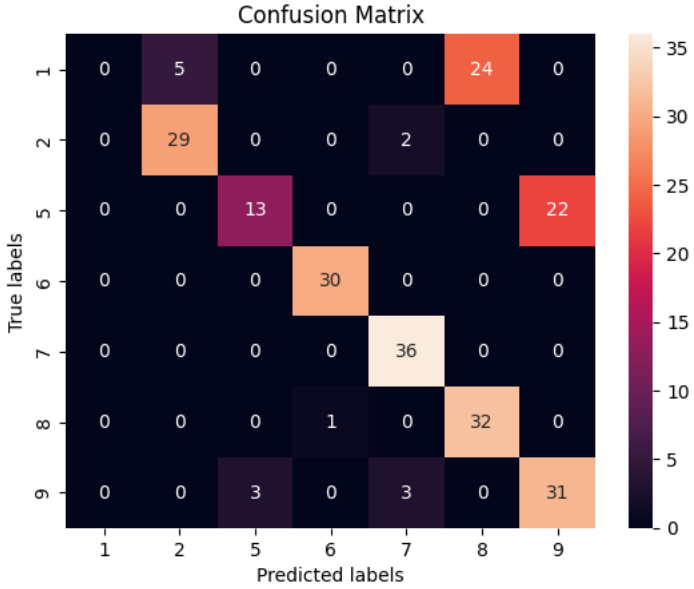

```
Label propagation model: 15 labeled & 315 unlabeled points (330 total)
Accuracy:  55.9%
Confusion matrix
[[ 0  5  0  0  0 24  0]
 [ 0 29  0  0  2  0  0]
 [ 0  0 13  0  0  0 22]
 [ 0  0  0 35  0  0  0]
 [ 0  0  0  0 36  0  0]
 [ 0  0  0  1  0 32  0]
 [ 0  0  3  0  3  0 31]]
```



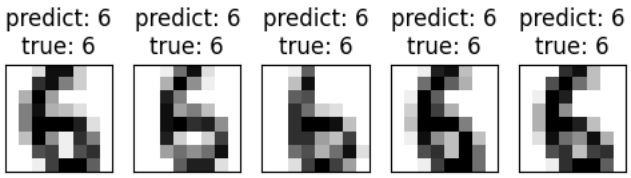
```
/usr/local/lib/python3.10/dist-packages/scipy/stats/_entropy.py:133: RuntimeWarning: invalid value encountered in divide
  pk = 1.0*pk / np.sum(pk, axis=axis, keepdims=True)
```



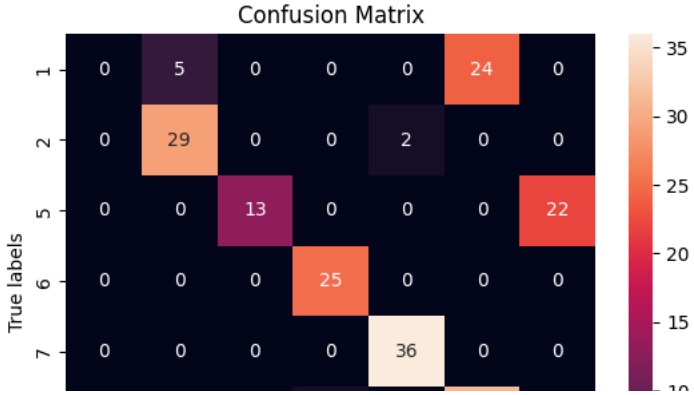

```
Label propagation model: 20 labeled & 310 unlabeled points (330 total)
Accuracy:  55.2%
Confusion matrix
[[ 0  5  0  0  0 24  0]
 [ 0 29  0  0  2  0  0]
 [ 0  0 13  0  0  0 22]
 [ 0  0  0 30  0  0  0]
 [ 0  0  0  0 36  0  0]
 [ 0  0  0  1  0 32  0]
 [ 0  0  3  0  3  0 31]]
```

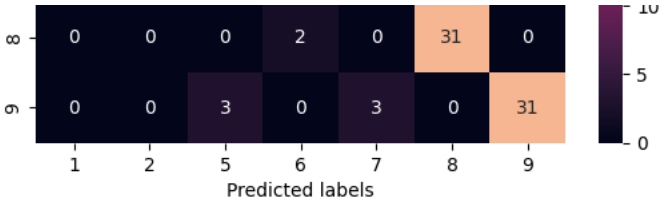



```
/usr/local/lib/python3.10/dist-packages/scipy/stats/_entropy.py:133: RuntimeWarning: invalid value encountered in divide
  pk = 1.0*pk / np.sum(pk, axis=axis, keepdims=True)
```

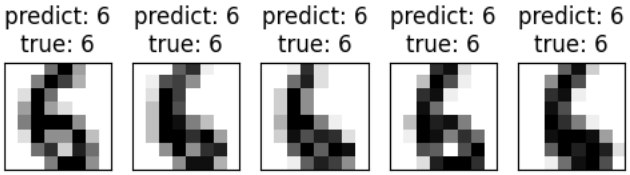



```
Label propagation model: 25 labeled & 305 unlabeled points (330 total)
Accuracy:  54.1%
Confusion matrix
[[ 0  5  0  0  0 24  0]
 [ 0 29  0  0  2  0  0]
 [ 0  0 13  0  0  0 22]
 [ 0  0  0 25  0  0  0]
 [ 0  0  0  0 36  0  0]
 [ 0  0  0  2  0 31  0]
 [ 0  0  3  0  3  0 31]]
```






```
/usr/local/lib/python3.10/dist-packages/scipy/stats/_entropy.py:133: RuntimeWarning: invalid value encountered in divide
  pk = 1.0*pk / np.sum(pk, axis=axis, keepdims=True)
```



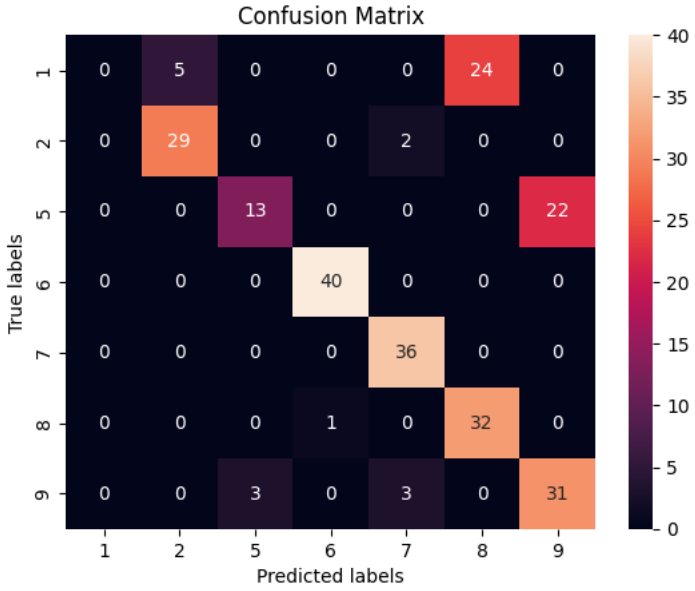

```
Label propagation model: 30 labeled & 300 unlabeled points (330 total)
Accuracy:  53.3%
Confusion matrix
[[ 0  5  0  0  0 24  0]
 [ 0 29  0  0  2  0  0]
```

The aforementioned analysis indicates that opting for high-accuracy labels negatively impacts the overall accuracy.

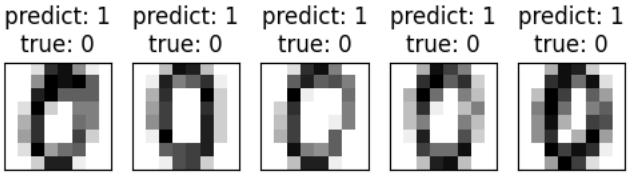
Subsequently, we proceed with a comparable exercise, this time selecting labels with low confidence, and assess the outcomes.

```
# Again Generate data with 330 images of digits because variables needs to be reintalized.
digits = datasets.load_digits()
rng = np.random.RandomState(0)
indices = np.arange(len(digits.data))
rng.shuffle(indices)

X = digits.data[indices[:330]]
y = digits.target[indices[:330]]
images = digits.images[indices[:330]]

n_total_samples = len(y)
n_labeled_points = 10
unlabeled_data_set = np.arange(n_total_samples)[n_labeled_points:]

# Learning algorithm with low confidence predicted labels
for i in range(5):
    y_train = np.copy(y)
    y_train[unlabeled_data_set] = -1

    Fu, labels, Y1 = LabelPropagation(X, y_train, tolerence=0.001, max_iter=300)

    predicted_labels = Fu[unlabeled_data_set]
    true_labels = y[unlabeled_data_set]

    PrintModel(true_labels, predicted_labels, labels, n_labeled_points)

    pred_entropies = stats.distributions.entropy(Y1.T)
    certain_index = np.argsort(pred_entropies)[::-1]
    certain_index = certain_index[np.in1d(certain_index, unlabeled_data_set)][:5]

    plotCertainNumbers(certain_index, Fu, y)

    delete_indices = np.array([], dtype=int)
    unlabeled_data_set = np.setdiff1d(unlabeled_data_set, certain_index)
    n_labeled_points+= len(certain_index)
```

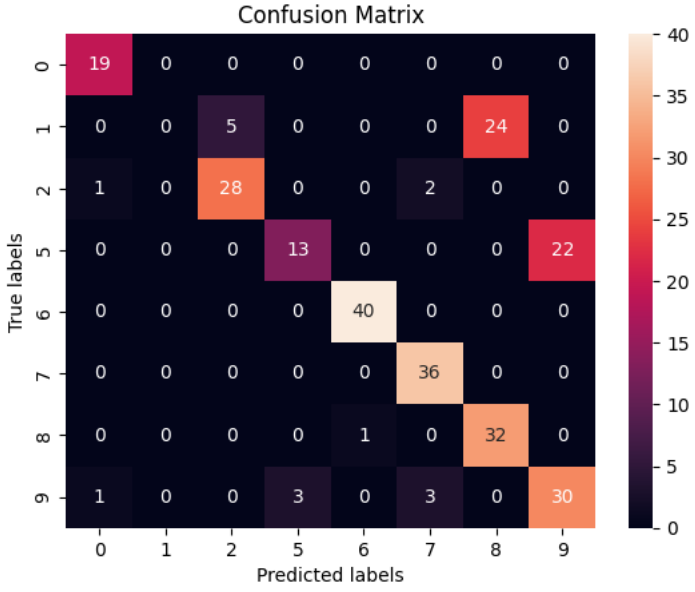
```
Label propagation model: 10 labeled & 320 unlabeled points (330 total)
Accuracy:  56.6%
Confusion matrix
[[ 0  5  0  0  0 24  0]
 [ 0 29  0  0  2  0  0]
 [ 0  0 13  0  0  0 22]
 [ 0  0  0 40  0  0  0]
 [ 0  0  0  0 36  0  0]
 [ 0  0  0  1  0 32  0]
 [ 0  0  3  0  3  0 31]]
```

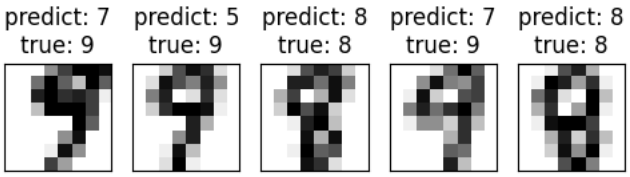



```
/usr/local/lib/python3.10/dist-packages/scipy/stats/_entropy.py:133: RuntimeW
  pk = 1.0*pk / np.sum(pk, axis=axis, keepdims=True)
```

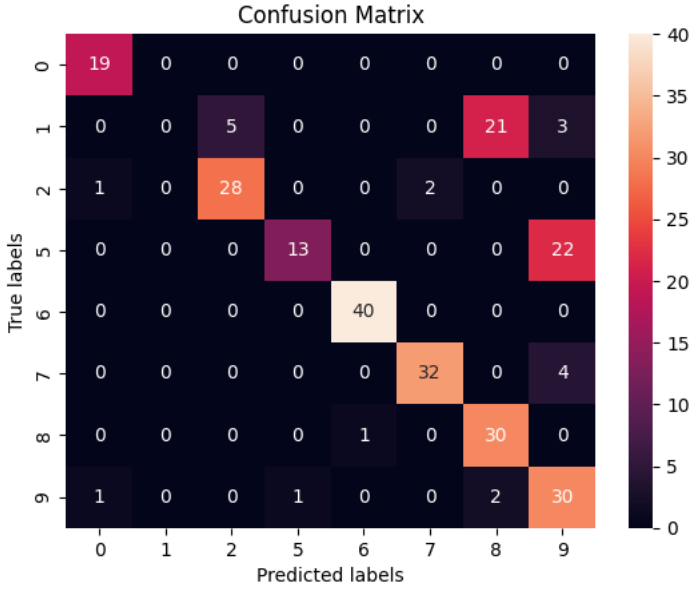


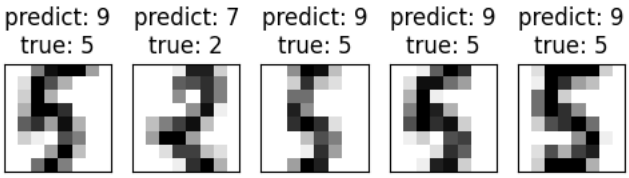

```
Label propagation model: 15 labeled & 315 unlabeled points (330 total)
Accuracy:  62.9%
Confusion matrix
[[19  0  0  0  0  0  0  0]
 [ 0  0  5  0  0  0 24  0]
 [ 1  0 28  0  0  2  0  0]
 [ 0  0  0 13  0  0  0 22]
 [ 0  0  0  0 40  0  0  0]
 [ 0  0  0  0  0 36  0  0]
 [ 0  0  0  0  1  0 32  0]
 [ 1  0  0  3  0  3  0 30]]
```



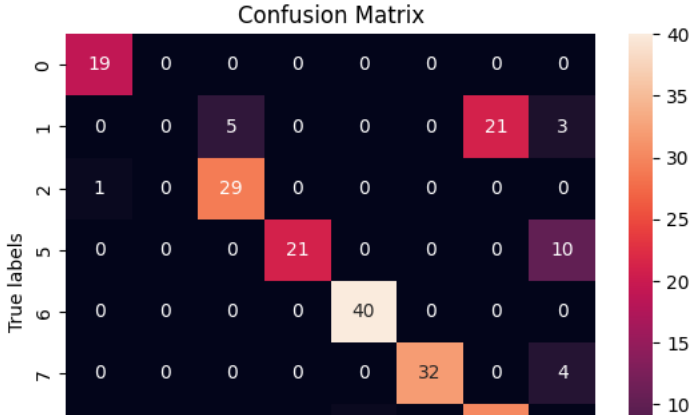


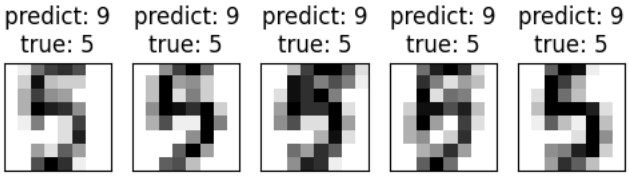

```
Label propagation model: 20 labeled & 310 unlabeled points (330 total)
Accuracy:  61.9%
Confusion matrix
[[19  0  0  0  0  0  0  0]
 [ 0  0  5  0  0  0 21  3]
 [ 1  0 28  0  0  2  0  0]
 [ 0  0  0 13  0  0  0 22]
 [ 0  0  0  0 40  0  0  0]
 [ 0  0  0  0  0 32  0  4]
 [ 0  0  0  0  1  0 30  0]
 [ 1  0  0  1  0  0  2 30]]
```



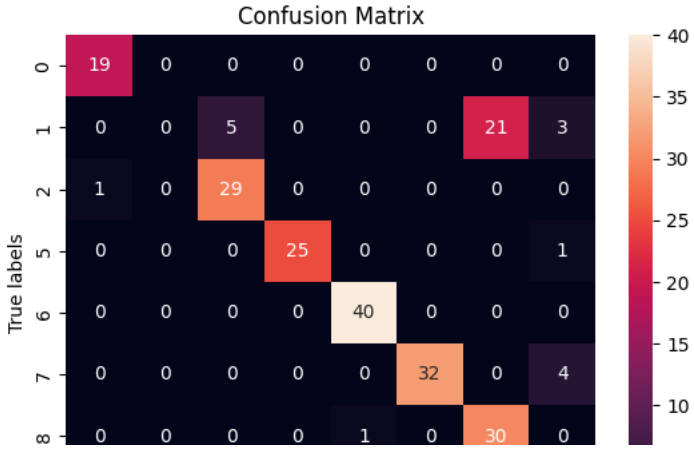



```
Label propagation model: 25 labeled & 305 unlabeled points (330 total)
Accuracy:  65.6%
Confusion matrix
[[19  0  0  0  0  0  0  0]
 [ 0  0  5  0  0  0 21  3]
 [ 1  0 29  0  0  0  0  0]
 [ 0  0  0 21  0  0  0 10]
 [ 0  0  0  0 40  0  0  0]
 [ 0  0  0  0  0 32  0  4]
 [ 0  0  0  0  1  0 30  0]
 [ 1  0  0  2  0  0  2 29]]
```



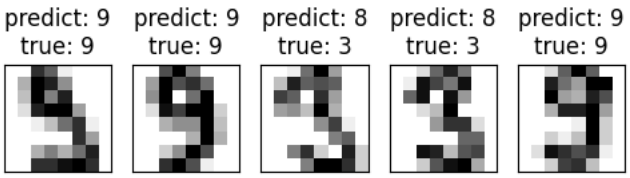



```
Label propagation model: 30 labeled & 300 unlabeled points (330 total)
Accuracy:  68.0%
Confusion matrix
[[19  0  0  0  0  0  0  0]
 [ 0  0  5  0  0  0 21  3]
 [ 1  0 29  0  0  0  0  0]
 [ 0  0  0 25  0  0  0  1]
 [ 0  0  0  0 40  0  0  0]
 [ 0  0  0  0  0 32  0  4]
 [ 0  0  0  0  1  0 30  0]
 [ 1  0  0  2  0  0  2 29]]
```



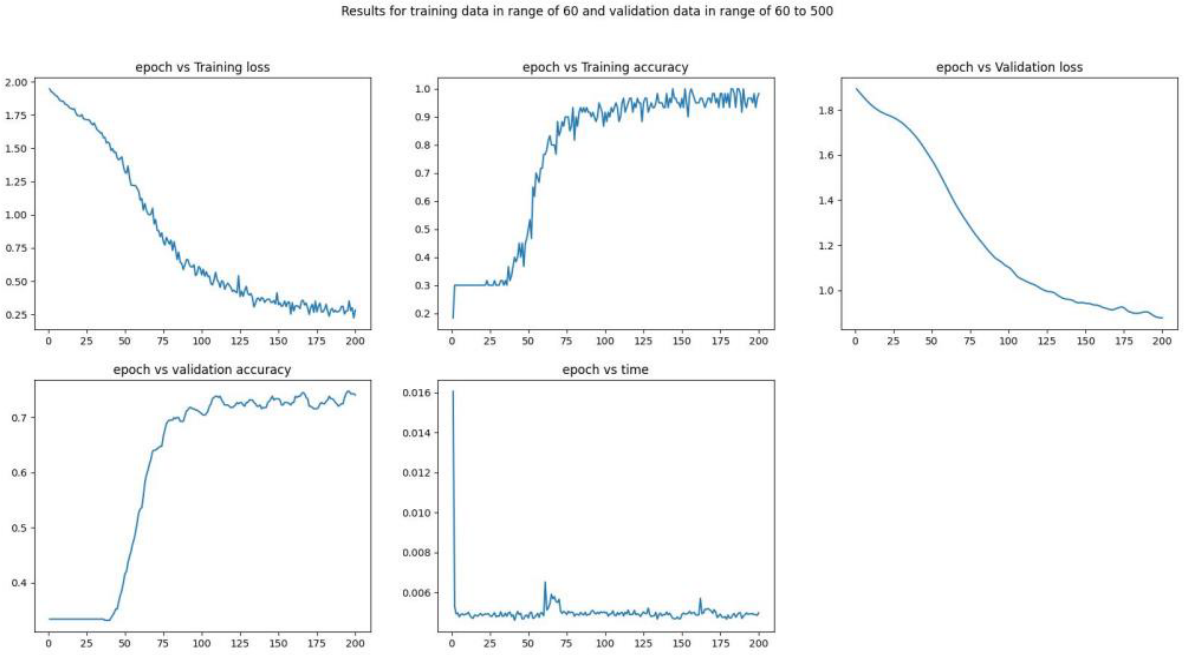
Conclusion:

The algorithm exhibits a favorable learning curve and achieves improved accuracy results.

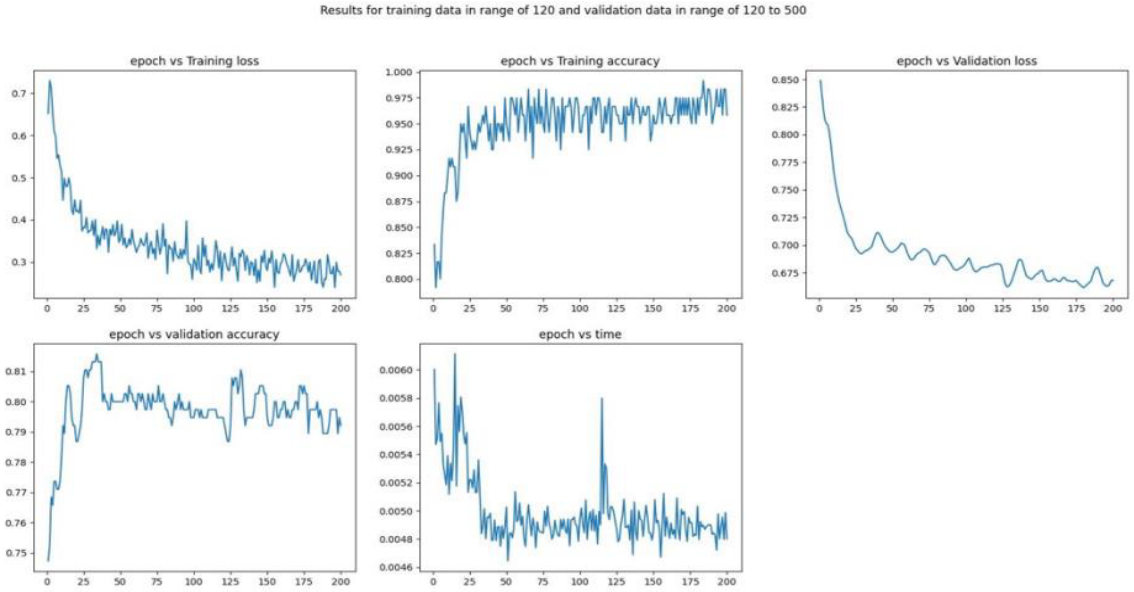


Solution 4

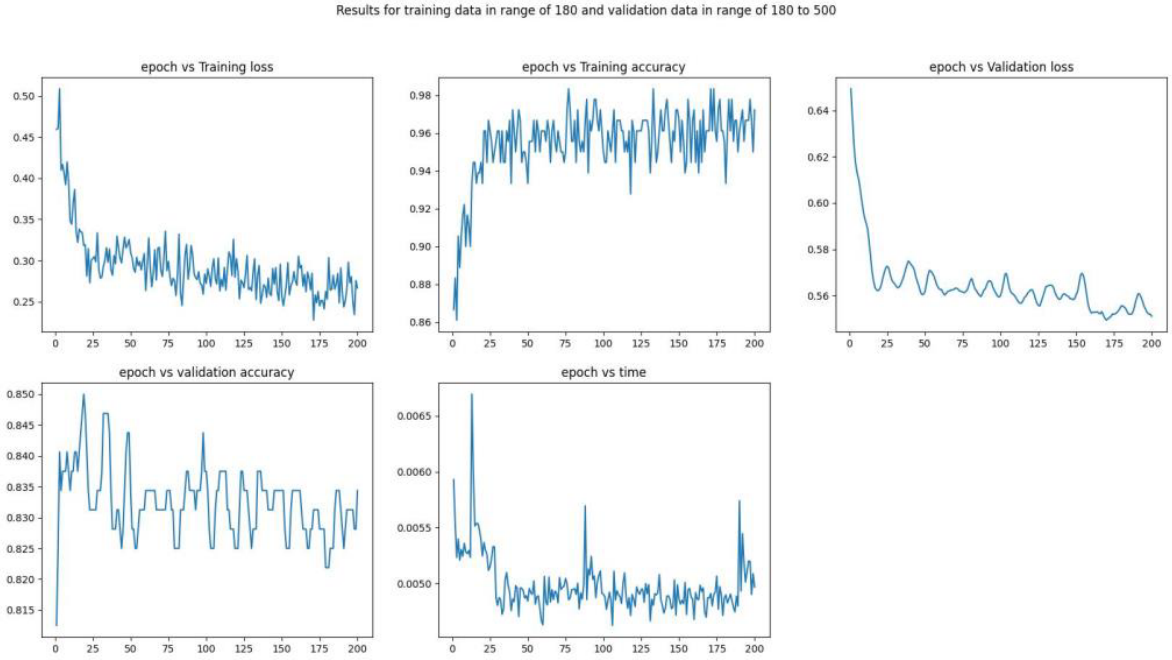
Labelled data of 60:



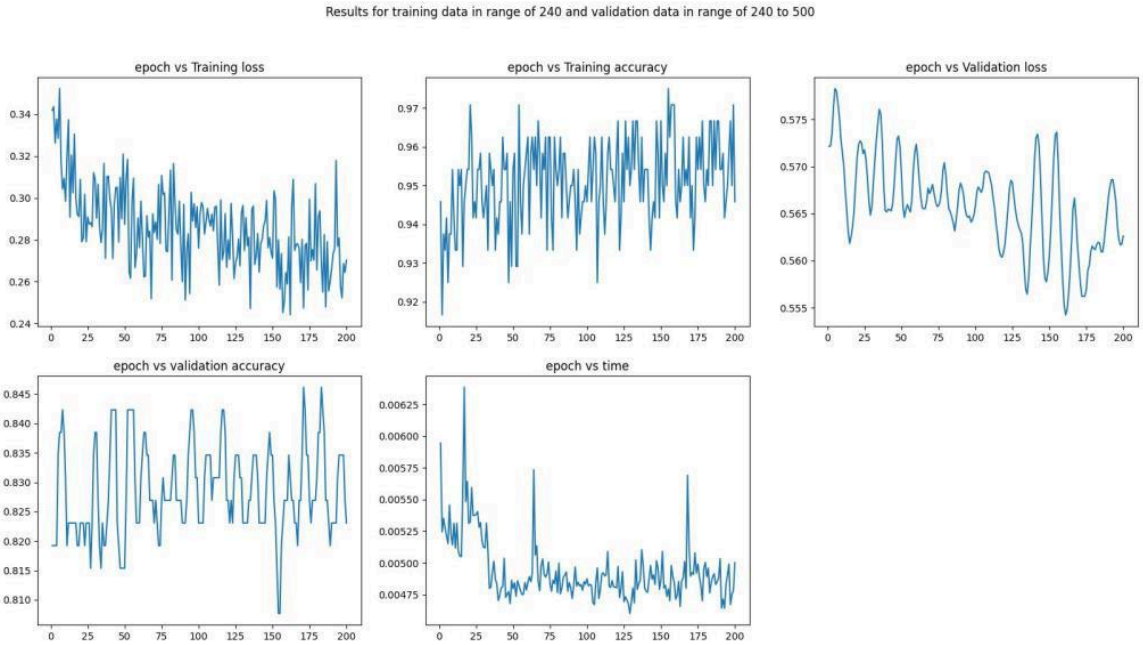
Labeled data of 120:



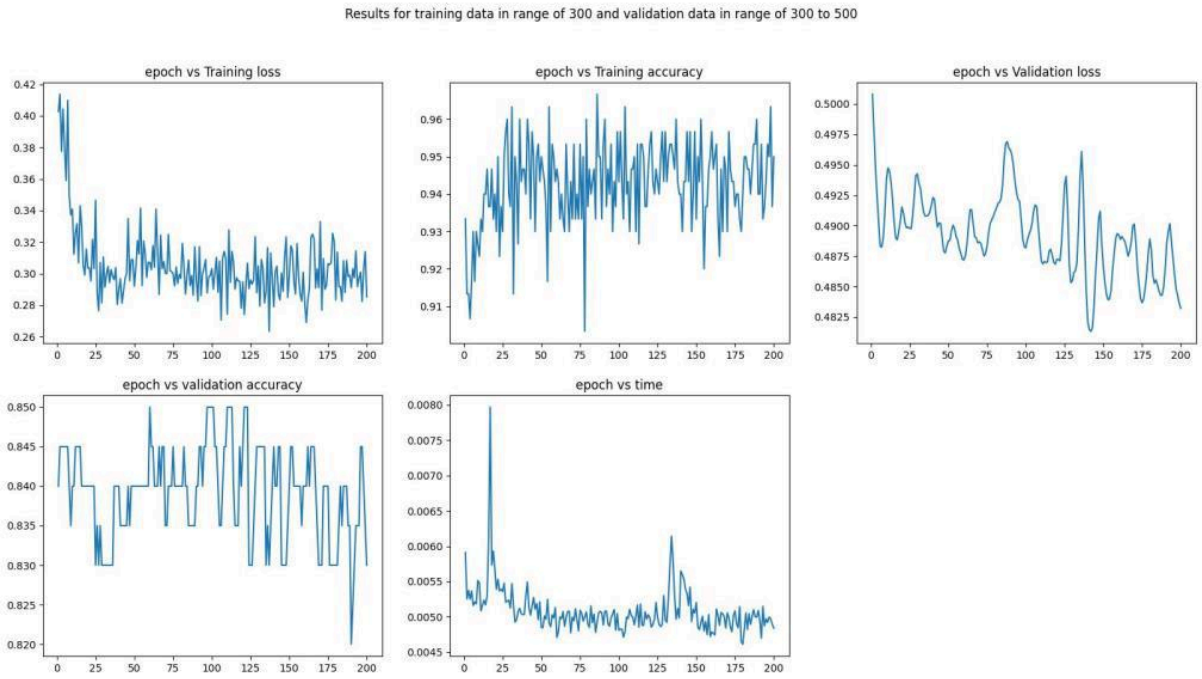
Labeled data of 180:



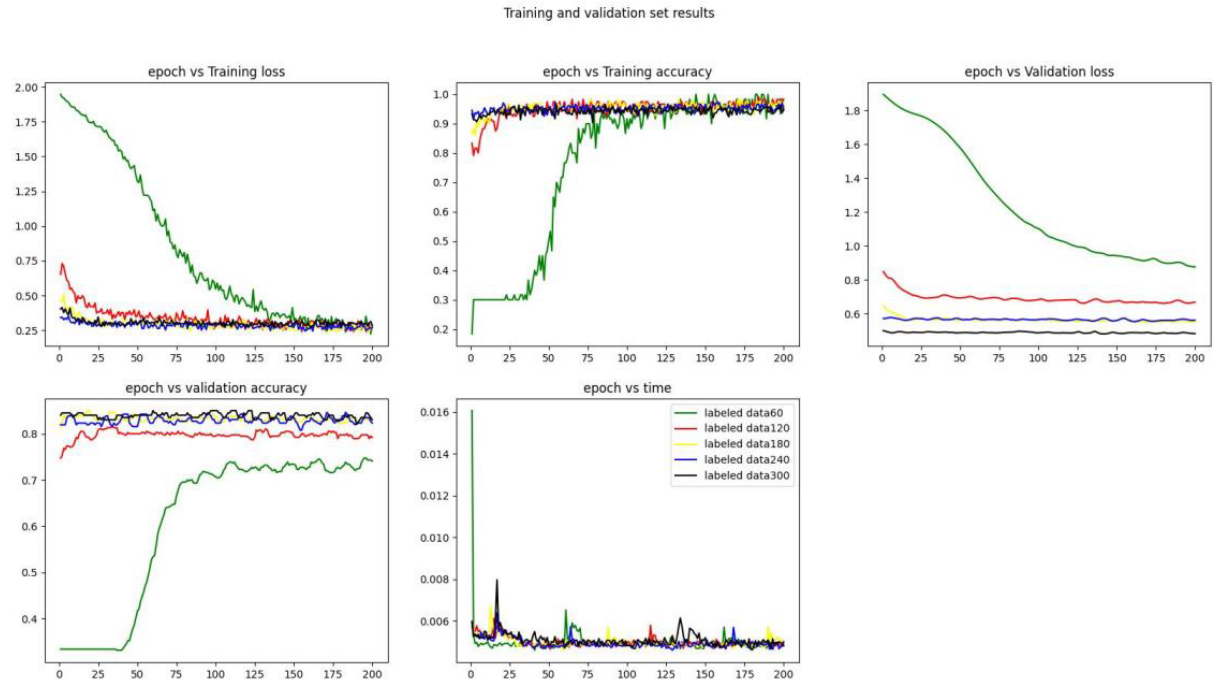
Labeled data of 240:



Labeled data of 300:



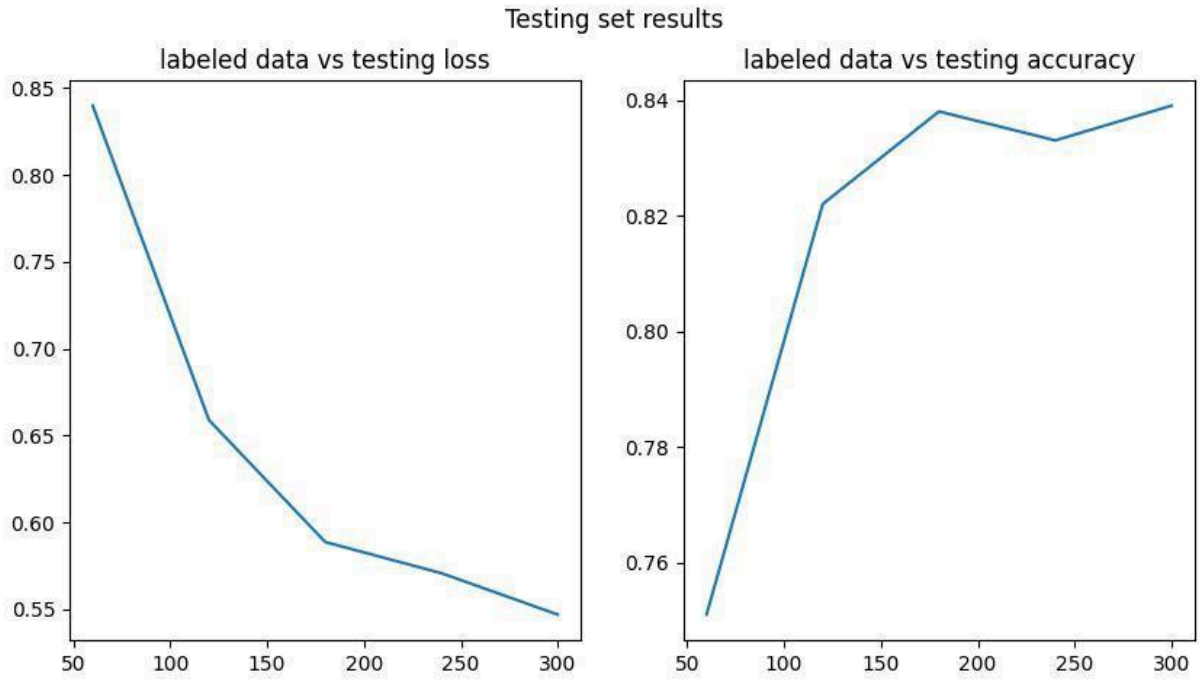
Combined Results:



Conclusion:

The above results indicate a positive correlation between the validation accuracy and the quantity of labeled data. Following a certain number of iterations, all models have achieved a commendable level of training accuracy.

Testing set results:



The testing accuracy exhibits an upward trend with an increasing volume of labeled data, reaching approximately 83% as the labeled data expands.

The test outcomes for labeled data corresponding to the numbers [60, 120, 180, 240, 300] are presented below.

```
Loading cora dataset...
Test set results: loss= 0.8398 accuracy= 0.7510
Test set results: loss= 0.6588 accuracy= 0.8220
Test set results: loss= 0.5886 accuracy= 0.8380
Test set results: loss= 0.5707 accuracy= 0.8330
Test set results: loss= 0.5469 accuracy= 0.8390
```