M12-HW1

November 30, 2024

1 Problem 1

1.1 Problem Description

In this problem you will use PCA and TSNE to apply dimensionality reduction to 64x64 images of signed distance fields (SDFs) on parts belonging to 8 different classes. Each class is topologicaly similar, with some variation in void size and shape. These signed distance fields are helpful in the prediction of internal stress fields in the parts. You will also apply KNN to predict the class of the part with the reduced space.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

- 3x8 subplot visualization of the first 3 samples from each of the 8 classes
- Bar plot of the variance explained for the first 25 PCs and the number of PCs required to explain $> 90\%$ of the variance in the training data
- 4x8 subplot visualization of reconstructed samples using 3, 10, 50 and all PCs on the first sample from each of the 8 classes in the test set
- Test accuracy of KNN classifier trained on the 3D, 10D, and 50D PCA reduced feature spaces
- Plot of the 2D TSNE reduced feature space
- Test accuracy of the KNN classifier trained on the 2D TSNE reduced feature space
- Discussion questions 1 and 2

Imports and Utility Functions:

[100]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import io

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

def dataLoader(filepath):
    # Load and flatten the SDF dataset
    mat = io.loadmat(filepath)
    data = []
    for i in range(800):
        sdf = mat["sdf"][i][0].T
        data.append(sdf.flatten())
    data = np.vstack(data)
    # Assign labels
    labels = np.repeat(np.arange(8), 100)
    return data, labels

def plot_sdf(data, ax = None, title = None):
    # If no axes, make them
    if ax is None:
        ax = plt.gca()
    # Reshape image data into square
    sdf = data.reshape(64,64)
    # Plot image, with bounds of the SDF values for the entire dataset
    ax.imshow(sdf, vmin=-0.31857, vmax=0.206349, cmap="jet")
    ax.axis('off')
    # If there is a title, add it
    if title:
        ax.set_title(title)
```

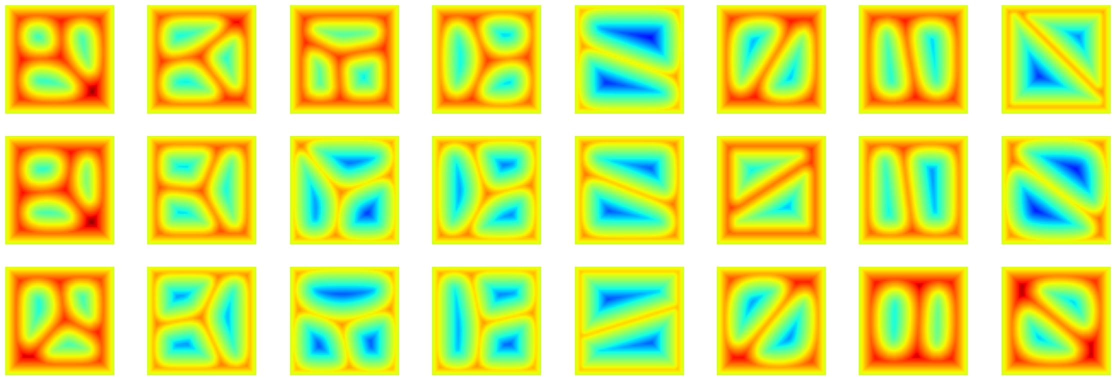
1.2 Visualization

Using the provided `dataLoader()` function, load the data and labels from `sdf_images.mat`. The returned data will contain 800 samples, with 4096 features. Then, using the provided `plot_sdf()` function, generate a 3x8 subplot figure containing visualizations of the first 3 SDFs in each class.

[101]:
```
# YOUR CODE GOES HERE
data,labels = dataLoader(r'C:\Users\zsqu4\Desktop\ML␣
 ↪HW\ML-for-engineers\HW12\HW12\data\sdf_images.mat')
# print(X)
# print(y)
print(data.shape)
print(labels.shape)

fig, ax = plt.subplots(3, 8, figsize=(15, 5), sharey=True)
for i in range(0,8):
    for j in range(3):
        plot_sdf(data[i*100+j],ax[j][i])
```

```
(800, 4096)
(800,)
```



1.3 Explained Variance

Use `train_test_split()` to partition the data and labels into a training and test set with `test_size = 0.2` and `random_state = 0`. Then train a PCA model on the training data and generate a bar plot of the variance explained for the first 25 principal components. Determine the number of principal components required to explain > 90% of the variance in the training data.

[102]:
```
# YOUR CODE GOES HERE
X_train,X_test,y_train,y_test = train_test_split(data,labels,test_size = 0.2,␣
↪random_state = 0)

pca = PCA(n_components=25)
pca.fit(X_train)

explained_variance = pca.explained_variance_ratio_

cumulative_variance = np.cumsum(explained_variance)
n_components_90 = np.argmax(cumulative_variance >= 0.90) + 1
print(n_components_90)
```

```
9
```

1.4 PCA Reconstruction

Using the training data, generate 4 PCA models using 3, 10, 50, and all of the principal components. Use these models to transform the test data into the reduced space, and then reconstruct the data from the reduced space. Plot the reconstruction for each model, on the first occurence of each class in the test set. Your generated plot should be a 4x8 subplot figure, with each subplot title containing the class and the number of PCs used.

[103]:
```
# YOUR CODE GOES HERE


max_components = min(X_train.shape[0], X_train.shape[1])
```

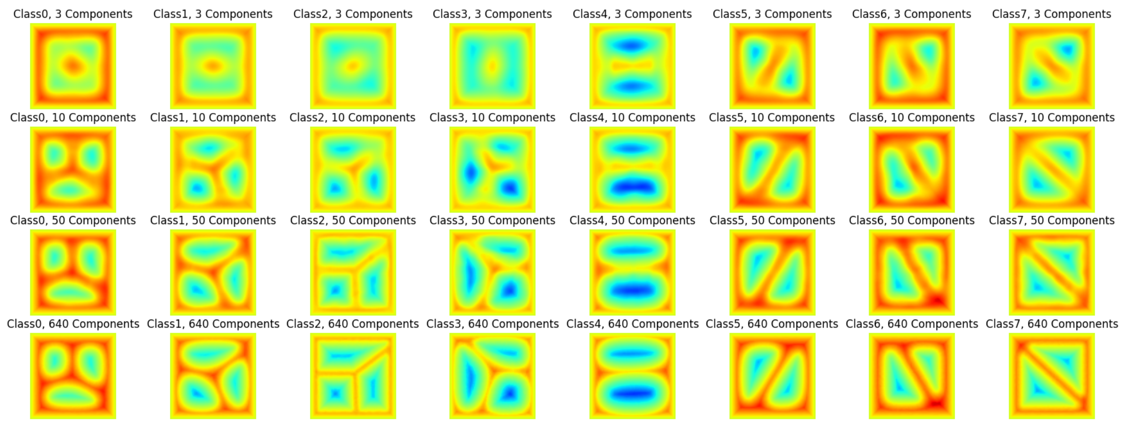
```
nums = [3,10,50,max_components]
classes = np.unique(y_test)

fig, axes = plt.subplots(len(nums),len(classes),figsize=(22, 8))


for j, num in enumerate(nums):
    # Train PCA on the training data
    pca = PCA(n_components=num)
    pca.fit(X_train)

    # Transform and reconstruct test data
    X_test_reduced = pca.transform(X_test)
    X_test_reconstructed = pca.inverse_transform(X_test_reduced)

    for i, cls in enumerate(classes):
        index = np.where(y_test== cls)[0][0]
        recon = X_test_reconstructed[index]
        plot_sdf(recon,axes[j][i],title = f'Class{cls}, {num} Components')
```



1.5 KNN on PCA Reduced Data

Now train a KNN classifier to predict the class of the 3D, 10D, and 50D PCA reduced data. You should train the KNN on the reduced training data, and report the prediction accuracy on the test set. You will also need to determine the `n_neighbors` parameter for your KNN classifier that gives good results.

[104]:
```
# YOUR CODE GOES HERE
nums = [3,10,50]


for num in nums:
    pca = PCA(n_components=num)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)
    best_accuracy = 0
    best_num = 0
    best_n = 0

    for n in range(1,100):

        knn = KNeighborsClassifier(n_neighbors=n)
        knn.fit(X_train_pca,y_train)
        preds = knn.predict(X_test_pca)
        accuracy = np.sum(preds == y_test)/y_test.shape[0]
        print('Accurcy',accuracy*100,'%')
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_num = num
            best_n = n



print("Best number of components",best_num,', Best number of␣
 ↪neighbors',best_n,", Best_accuracy",best_accuracy,)
```

```
Accurcy 68.125 %
Accurcy 67.5 %
Accurcy 70.625 %
Accurcy 70.625 %
Accurcy 70.0 %
Accurcy 70.0 %
Accurcy 71.25 %
Accurcy 69.375 %
Accurcy 70.0 %
Accurcy 69.375 %
Accurcy 71.25 %
Accurcy 71.25 %
Accurcy 71.25 %
Accurcy 71.25 %
Accurcy 70.625 %
Accurcy 71.875 %
Accurcy 72.5 %
Accurcy 71.25 %
Accurcy 71.25 %
Accurcy 72.5 %
Accurcy 72.5 %
Accurcy 69.375 %
Accurcy 72.5 %
Accurcy 72.5 %
Accurcy 71.875 %
Accurcy 72.5 %
Accurcy 70.0 %
Accurcy 70.625 %
Accurcy 72.5 %
Accurcy 73.125 %
Accurcy 71.25 %
Accurcy 73.125 %
Accurcy 72.5 %
Accurcy 72.5 %
Accurcy 72.5 %
Accurcy 72.5 %
Accurcy 72.5 %
Accurcy 72.5 %
Accurcy 71.25 %
Accurcy 69.375 %
Accurcy 70.625 %
Accurcy 68.75 %
Accurcy 68.125 %
Accurcy 68.75 %
Accurcy 69.375 %
Accurcy 71.25 %
Accurcy 71.25 %
Accurcy 71.875 %
Accurcy 72.5 %
Accurcy 71.875 %
Accurcy 70.625 %
Accurcy 71.25 %
Accurcy 71.25 %
Accurcy 71.875 %
Accurcy 73.125 %
Accurcy 73.75 %
Accurcy 71.875 %
Accurcy 71.25 %
Accurcy 70.625 %
Accurcy 70.0 %
Accurcy 68.75 %
Accurcy 68.125 %
Accurcy 67.5 %
Accurcy 66.875 %
Accurcy 66.25 %
Accurcy 68.125 %
Accurcy 70.0 %
Accurcy 69.375 %
Accurcy 68.75 %
Accurcy 70.0 %
Accurcy 70.0 %
Accurcy 68.125 %
Accurcy 70.625 %
Accurcy 70.0 %
Accurcy 70.0 %
Accurcy 69.375 %
Accurcy 68.125 %
Accurcy 66.25 %
Accurcy 66.875 %
Accurcy 66.25 %
Accurcy 65.0 %
Accurcy 66.875 %
Accurcy 66.875 %
Accurcy 66.25 %
Accurcy 64.375 %
Accurcy 64.375 %
Accurcy 64.375 %
Accurcy 64.375 %
Accurcy 61.875 %
Accurcy 61.875 %
Accurcy 61.875 %
Accurcy 61.875 %
Accurcy 61.875 %
Accurcy 63.74999999999999 %
Accurcy 63.74999999999999 %
Accurcy 62.5 %
Accurcy 62.5 %
Accurcy 62.5 %
Accurcy 63.125 %
Accurcy 89.375 %
Accurcy 85.625 %
Accurcy 86.25 %
Accurcy 86.25 %
Accurcy 86.875 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 91.25 %
Accurcy 90.625 %
Accurcy 91.25 %
Accurcy 90.625 %
Accurcy 91.875 %
Accurcy 90.625 %
Accurcy 91.25 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 91.875 %
Accurcy 91.875 %
Accurcy 91.25 %
Accurcy 91.25 %
Accurcy 91.25 %
Accurcy 92.5 %
Accurcy 91.875 %
Accurcy 91.875 %
Accurcy 91.875 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 88.75 %
Accurcy 91.25 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 88.75 %
Accurcy 88.125 %
Accurcy 88.125 %
Accurcy 88.75 %
Accurcy 88.125 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 88.75 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 87.5 %
Accurcy 88.75 %
Accurcy 86.875 %
Accurcy 88.125 %
Accurcy 88.125 %
Accurcy 89.375 %
Accurcy 91.25 %
Accurcy 90.625 %
Accurcy 91.25 %
Accurcy 90.625 %
Accurcy 91.875 %
Accurcy 90.625 %
Accurcy 92.5 %
Accurcy 93.125 %
Accurcy 92.5 %
Accurcy 92.5 %
Accurcy 92.5 %
Accurcy 92.5 %
Accurcy 92.5 %
Accurcy 92.5 %
Accurcy 92.5 %
Accurcy 91.25 %
Accurcy 91.25 %
Accurcy 91.25 %
Accurcy 91.875 %
Accurcy 91.25 %
Accurcy 91.25 %
Accurcy 90.625 %
Accurcy 91.25 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 89.375 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 88.75 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 89.375 %
Best number of components 50 , Best number of neighbors 15 , Best_accuracy
0.93125
```

1.6 TSNE Visualization

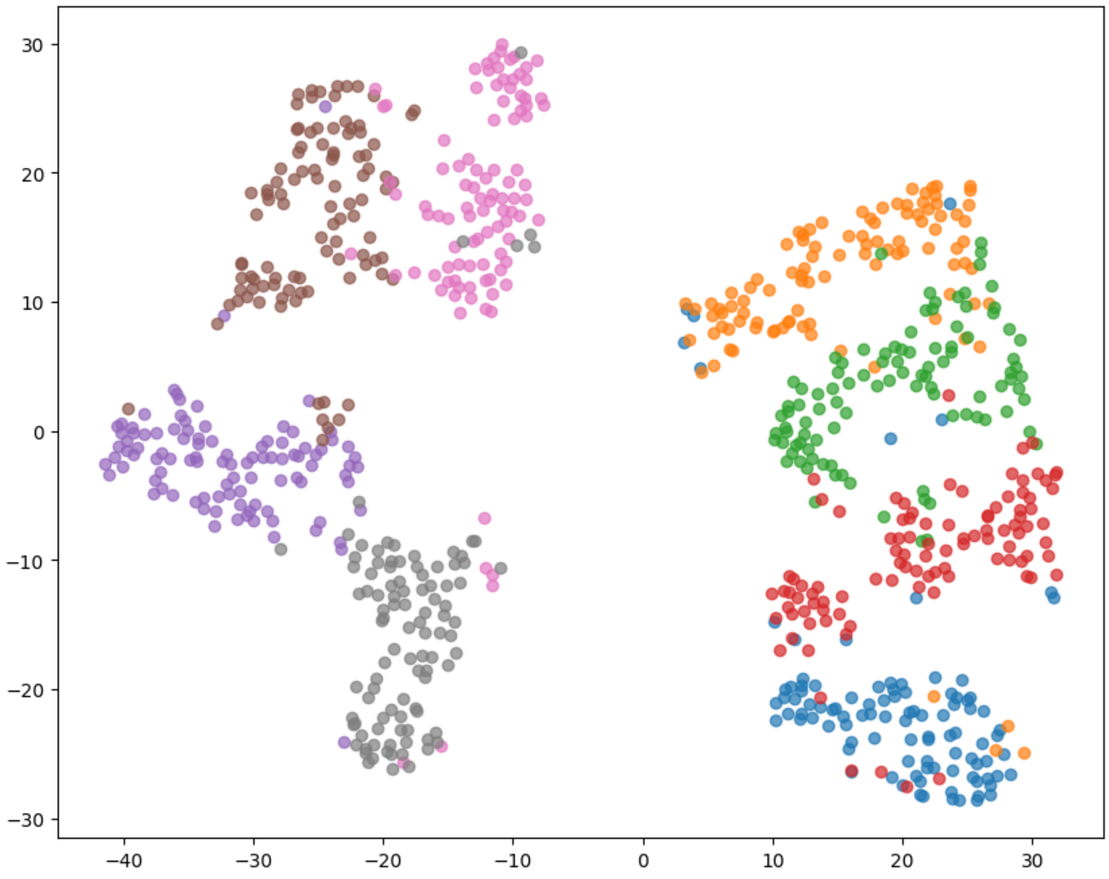
First reduced the full dataset to 50D using PCA, and then further reduced the data to 2D using TSNE. Plot the 2D reduced feature space with a scatter plot, coloring each point according to its class.

[105]:
```
# YOUR CODE GOES HERE
pca = PCA(n_components=50)
data_pca = pca.fit_transform(data)

tsne = TSNE(n_components = 2,learning_rate = 'auto')
```

```
data_tsne = tsne.fit_transform(data_pca)

plt.figure(figsize=(10, 8))
for class_label in np.unique(labels):
    indices = np.where(labels == class_label)
    plt.scatter(
        data_tsne[indices, 0],
        data_tsne[indices, 1],
        label=f"Class {class_label}",
        alpha=0.7
    )
```



1.7 KNN on PCA/TSNE Reduced Data

Using the same 2D PCA/TSNE data, split the data into train and test data and labels using `train_test_split` with a `random_state = 0` parameter so you have the same train/test partition as before. Then, train a KNN on this 2D feature space with the training set, and report the KNN classifier accuracy on the test set. Again, you will need to determine the `n_neighbors` parameter in the KNN classifier that gives good results.

```
[106]: # YOUR CODE GOES HERE
X_train,X_test,y_train,y_test = train_test_split(data_tsne,labels,test_size = 0.
 ↪2, random_state = 0)

best_accuracy = 0
best_num = 0
best_n = 0

for n in range(1,100):

    knn = KNeighborsClassifier(n_neighbors=n)
    knn.fit(X_train,y_train)
    preds = knn.predict(X_test)
    accuracy = np.sum(preds == y_test)/y_test.shape[0]
    print('Accurcy',accuracy*100,'%')
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_num = num
        best_n = n

print("Best number of components",best_num,', Best number of␣
 ↪neighbors',best_n,", Best_accuracy",best_accuracy,)
```

```
Accurcy 86.25 %
Accurcy 85.625 %
Accurcy 87.5 %
Accurcy 87.5 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 88.75 %
Accurcy 90.625 %
Accurcy 89.375 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 91.875 %
Accurcy 90.0 %
Accurcy 91.25 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 91.25 %
Accurcy 90.625 %
Accurcy 91.25 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.625 %
Accurcy 90.0 %
Accurcy 90.0 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.125 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 89.375 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 88.75 %
Accurcy 89.375 %
Accurcy 89.375 %
Accurcy 88.125 %
Accurcy 88.125 %
Accurcy 87.5 %
Accurcy 87.5 %
Accurcy 87.5 %
Accurcy 88.125 %
Accurcy 87.5 %
Accurcy 88.125 %
Accurcy 87.5 %
Accurcy 87.5 %
Accurcy 88.125 %
Accurcy 87.5 %
Accurcy 86.875 %
Accurcy 87.5 %
Accurcy 88.125 %
Accurcy 87.5 %
Accurcy 87.5 %
Accurcy 86.875 %
Accurcy 86.875 %
Accurcy 87.5 %
Accurcy 87.5 %
Best number of components 50 , Best number of neighbors 15 , Best_accuracy
0.91875
```

1.8 Discussion

1. Discuss how the number of principal components relates to the quality of reconstruction of the data. Using all of the principal components, should there be any error in the reconstruction of a sample from the training data? What about in the reconstruction of an unseen sample from the testing data?
2. Discuss how you determined `k`, the number of neighbors in your KNN models. Why do we perform dimensionality reduction to our data before feeding it to our KNN classifier?

Your response goes here

1. Generally, more principal components gives better quality of reconstruction. With all principal components used, there are still error between reconstructed data and original data.
2. I determined a range of k from 1-50. And I train the model with different parameter and compare their performances. We reduce the dimensions because high number of dimensionality brings noise and lowers computational efficiency