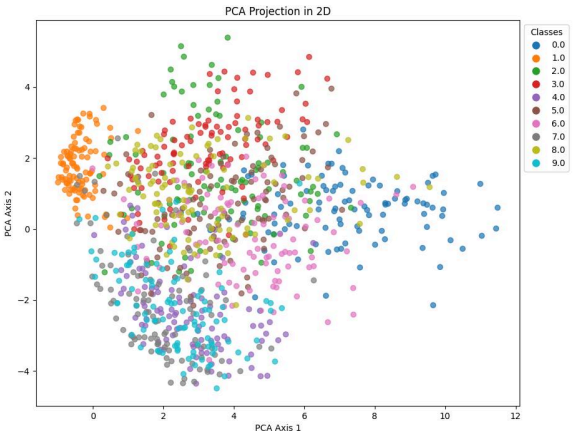
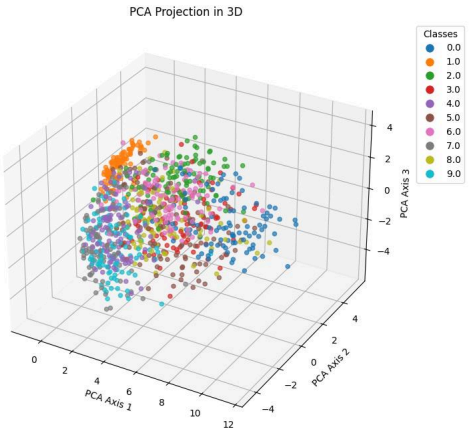
1.



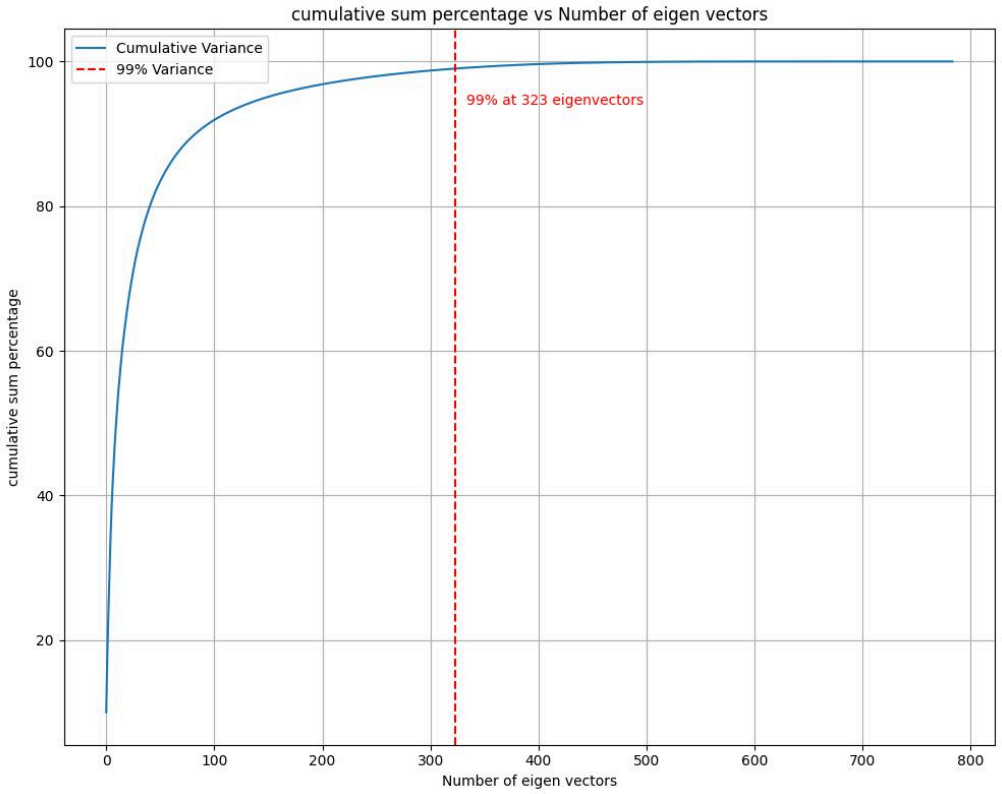



```
eVals_flip, eVecs_flip = pca(X, L)
projectedData2D, projectedData3D = eigenProjection(eVecs_flip)
samplesNeeded = 100
sampleSelctionNplots(projectedData2D, projectedData3D, uniqueClss, samplesNeeded, L, "PCA")
```

```
def eigenProjection(eVecs):
    # Project data into 2D and 3D space
    projectedData2D = np.matmul(X, eVecs[:, :2])
    projectedData3D = np.matmul(X, eVecs[:, :3])
    return projectedData2D, projectedData3D
```

```
def sampleSelctionNplots(projectedData2D, projectedData3D, uniqueClss, samplesNeeded, L, name):
    smallSamples = []
    for digit in uniqueClss:
        indices = np.where(L == digit)[0]
        minNum = min(samplesNeeded, len(indices))
        randomClsSample = random.sample(list(indices), minNum)
        smallSamples.extend(randomClsSample)

    projectedData2D = projectedData2D[smallSamples]
    projectedData3D = projectedData3D[smallSamples]
    L = L[smallSamples]

    # 2D Scatter Plot
    plt.figure(figsize=(10, 8))
    distribution2D = plt.scatter(projectedData2D[:, 0], projectedData2D[:, 1], c=L, cmap='tab10', alpha=0.7, marker='o')
    colors = distribution2D.cmap(distribution2D.norm(np.unique(L)))
    handles = [
        plt.Line2D([0], [0], marker='o', color=color, linestyle='', markersize=8, label=str(label))
        for color, label in zip(colors, np.unique(L))
    ]
    plt.legend(handles=handles, title="Classes", bbox_to_anchor=(1, 1), loc='upper left', frameon=True)
    plt.xlabel(f"{name} Axis 1")
    plt.ylabel(f"{name} Axis 2")
    plt.title(f"{name} Projection in 2D")
    plt.savefig(f"{name}2D.jpeg")
    plt.show()
    plt.clf()

    # 3D Scatter Plot
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    distribution3D = ax.scatter(projectedData3D[:, 0], projectedData3D[:, 1], projectedData3D[:, 2], c=L, cmap='tab10', alpha=0.7)
    colors = distribution3D.cmap(distribution3D.norm(np.unique(L)))
    ax.set_xlabel(f"{name} Axis 1")
    ax.set_ylabel(f"{name} Axis 2")
    ax.set_zlabel(f"{name} Axis 3")
    ax.set_title(f"{name} Projection in 3D")
    handles = [
        plt.Line2D([0], [0], marker='o', color=color, linestyle='', markersize=8, label=str(label))
        for color, label in zip(colors, np.unique(L))
    ]
    ax.legend(handles=handles, title="Classes", bbox_to_anchor=(1.05, 1), loc='upper left', frameon=True)
    plt.savefig(f"{name}3D.jpeg")
    plt.show()
    plt.clf()
```

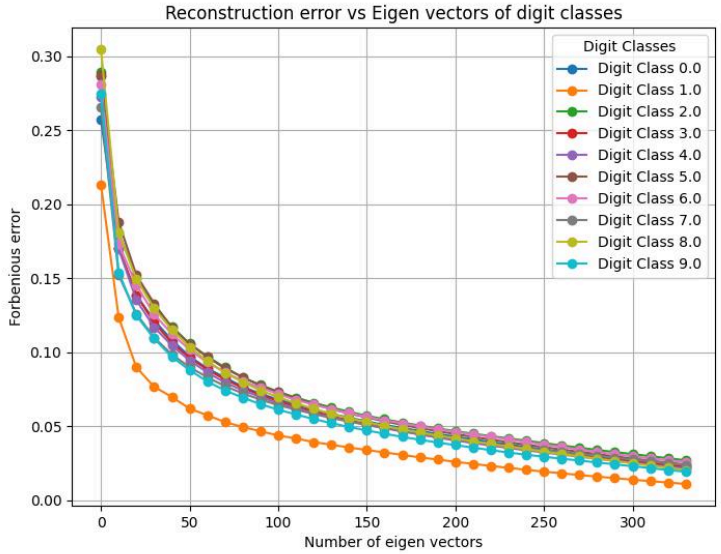
2. **Total eigenvectors** are needed to capture the 99% of the total variance of the data is **323**




```
eVals_flip, eVecs_flip = pca(X, L)
projectedData2D, projectedData3D = eigenProjection(eVecs_flip)
samplesNeeded = 100
sampleSelctionNplots(projectedData2D, projectedData3D, uniqueClss, samplesNeeded, L, "PCA")
totalVariance = np.sum(eVals_flip)
eVals_flip = eVals_flip/totalVariance
cumsumeVal = np.cumsum(eVals_flip)
vecs = np.argmax(cumsumeVal >= 0.99) + 1
print("Total eigenvectors are needed to capture the 99% of the total variance of the data is ",  vecs)
plt.plot(range(len(cumsumeVal)), cumsumeVal)
plt.xlabel(f"Number of eigen vectors")
plt.ylabel(f"cumulative sum")
plt.title(f"cumulative sum vs Number of eigen vectors")
plt.savefig(f"cumulative sum vs Number of eigen vectors.jpeg")
```

Comment: We don't need all the eigenvectors to account for maximum variance in the lower dimension. The first few eigenvectors account for maximum variance, which can result in slight information loss. This calculative tradeoff is better than using computation-heavy datasets.

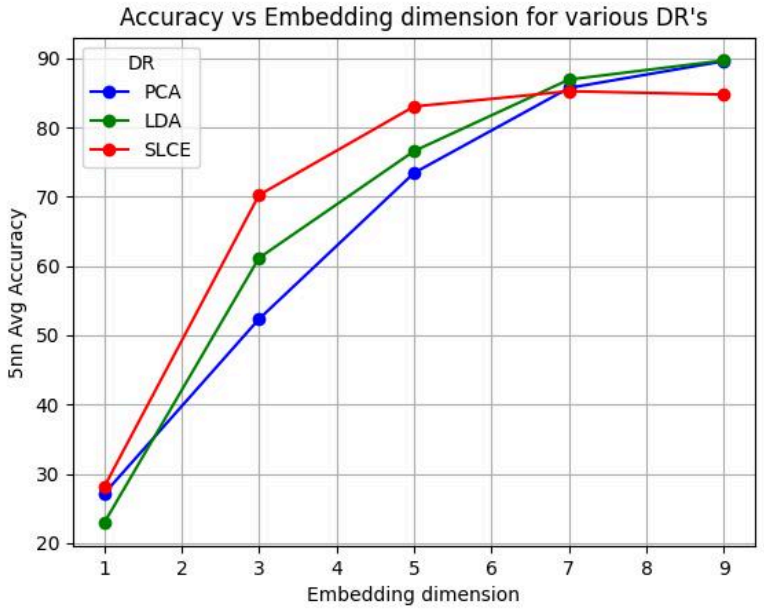
Average Reconstruction Error - Forbenious is used for Reconstruction loss



```
for i, cls in enumerate(uniqueClss):
    indices = np.where(L == cls)[0]
    clsList = X[indices, :]
    n = clsList.shape[0]
    errorVal = []
    for eigenVector in eigenVecList:
        eigenVector = eigenVector + 1
        eVecsTotalVariance = eVecs_flip[:, :eigenVector]
        newX = np.matmul(clsList, eVecsTotalVariance)
        reconstructedX = np.dot(newX, eVecsTotalVariance.T)
        fNorm = reconstructedX - clsList
        reconstructionError = np.linalg.norm(fNorm) / n
        errorVal.append(reconstructionError)

    name = "Digit Class " + str(cls)
    plt.plot(eigenVecList, errorVal, color = colors[i % 10], label=name, marker='o')
plt.xlabel(f"Number of eigen vectors")
plt.ylabel(f"Forbenious error")
plt.title(f"Reconstruction error vs Eigen vectors of digit classes")
plt.legend(loc="upper right", title="Digit Classes")
plt.grid(True)
plt.savefig(f"Reconstruction error of digit classes.jpeg")
# plt.show()
plt.clf()
```

Conclusion: Reconstruction error is decreasing as more and more eigenvectors are being added, which can be inferred that only the first few eigenvectors hold the most important information.

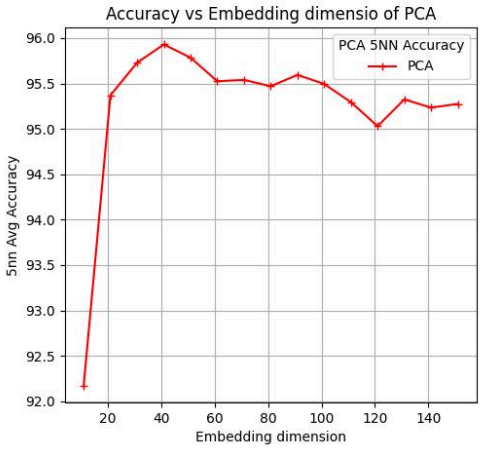
3.

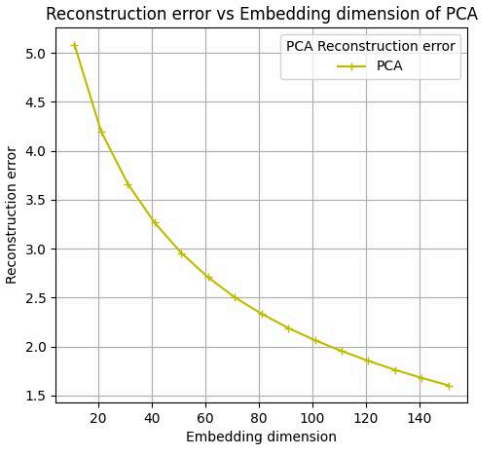



```
def accCal(eVecsPCA, embeddDim, itr):
    knn = KNeighborsClassifier(n_neighbors=5)
    avgAcc = []
    for p in embeddDim:
        accList = []
        eVecsTotalVariance = eVecsPCA[:, :p]
        newX = np.matmul(X, eVecsTotalVariance)
        for i in range(itr):
            Xtrain, Xtest, Ltrain, Ltest = train_test_split(newX, L, test_size=0.2, stratify=L, random_state=None)
            Ltrain = Ltrain.ravel()
            knn = KNeighborsClassifier(n_neighbors=5)
            knn.fit(Xtrain, Ltrain)
            LPred = knn.predict(Xtest)
            acc = accuracy_score(Ltest, LPred)
            accList.append(acc)
        pACC = np.mean(accList)
        avgAcc.append(pACC)
    avgAcc = np.array(avgAcc)*100
    return avgAcc


def knnACC(X, L):
    eValsPCA, eVecsPCA = pca(X, L)
    clsLenLDA, eValsLDA, eVecsLDA = embeddMatrix(X, L)
    eValsLDA, eVecsLDA = np.flip(eValsLDA), np.flip(eVecsLDA, axis = 1)
    eValsSLCE, eVecsSLCE = SLCE(X, L)
    embeddDim = list(range(1, 10, 2))
    avgAccPCA = accCal(eVecsPCA, embeddDim, 10)
    avgAccLDA = accCal(eVecsLDA, embeddDim, 10)
    avgAccSLCE = accCal(eVecsSLCE, embeddDim, 10)
    # colors = cm.tab10.colors
    plt.figure()
    plt.plot(embeddDim, avgAccPCA, color='b', marker='o', label = "PCA")
    plt.plot(embeddDim, avgAccLDA, color='g', marker='o', label = "LDA")
    plt.plot(embeddDim, avgAccSLCE, color='r', marker='o', label = "SLCE")
    plt.xlabel(f"Embedding dimension")
    plt.ylabel(f"5nn Avg Accuracy")
    plt.title(f"Accuracy vs Embedding dimension for various DR's")
    plt.legend(loc="upper left", title="DR")
    plt.grid(True)
    plt.savefig(f"Accuracy vs Embedding dimension.jpeg")
    # plt.show()
    plt.clf()
```

Conclusion: Almost 85 % or above accuracy is reached just at 9 eigenvectors. SLCE is powerful in very low dimensions but as dimensions increase, LDA and PCA take over. SLCE tries to preserve geometric structure but can perform low in comparison with LDA & PCA in high dimension.

PCA plots the reconstruction error and the 5NN accuracy as a function of embedding dimensions:






```python
def pcaReconstruction(X, L):
    eVals, eVecs = pca(X, L)
    embeddDim = list(range(11, 161, 10))
    errorVal = []
    for eigenVector in embeddDim:
        eigenVector = eigenVector + 1
        eVecsTotalVariance = eVecs[:, :eigenVector]
        newX = np.matmul(X, eVecsTotalVariance)
        reconstructedX = np.dot(newX, eVecsTotalVariance.T)
        fNorm = reconstructedX - X
        reconstructionError = np.linalg.norm(fNorm) / X.shape[0]
        errorVal.append(reconstructionError)

    errorVal = np.array(errorVal)*100
    avgAccPCA = accCal(eVecs, embeddDim, 10)
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))
    ax[0].plot(embeddDim, avgAccPCA, color='r', marker='+', label = "PCA")
    ax[0].set_xlabel(f"Embedding dimension")
    ax[0].set_ylabel(f"5nn Avg Accuracy")
    ax[0].set_title(f"Accuracy vs Embedding dimensio of PCA")
    ax[0].legend(loc="upper right", title="PCA 5NN Accuracy")
    ax[0].grid(True)
    # ax[0].savefig(f"PCA 5NN Accuracy vs Embedding dimension.jpeg")
    ax[1].plot(embeddDim, errorVal, color='y', marker='+', label = "PCA")
    ax[1].set_xlabel(f"Embedding dimension")
    ax[1].set_ylabel(f"Reconstruction error")
    ax[1].set_title(f"Reconstruction error vs Embedding dimension of PCA")
    ax[1].legend(loc="upper right", title="PCA Reconstruction error")
    ax[1].grid(True)


    fig.savefig("Reconstruction error of PCA.jpeg")
    plt.tight_layout()
    # plt.show()
    plt.clf()
```

Conclusion: As stated earlier, the first few eigenvectors hold the most important information. 99% variance is reached at 323 eigenvectors, but the max classification is reached at 40 eigenvectors, which again reiterates the above-discussed point. An increase in embedding dim does not guarantee bigger accuracy as the first few eigen vectors hold more important information.
The embedding matrix acts as a firm transformation matrix with more increase in embedding dimension which means an increase in more number of eigenvectors is more apt as it decreases the reconstruction error.