Nickhil Tekwani || CS6220 || HW 3a

PROBLEM 1: Supervised Classification Libraries: Regression, Decision Tree

In [ ]:
```
from sklearn.datasets import fetch_openml, fetch_20newsgroups
import pandas as pd

# Load MNIST
mnist = fetch_openml('mnist_784')
```

In [6]:
```
# Load 20NG
ng20 = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))
```

In [10]:
```
spambase = pd.read_csv('./spambase/spambase.data', header=None, delimiter=',')
spambase_data = spambase.iloc[:, :-1]  # All columns except the last one (featu
spambase_target = spambase.iloc[:, -1]  # The last column (label)
```

In [11]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler

# MNIST
mnist_data = StandardScaler().fit_transform(mnist.data)
mnist_target = mnist.target

# 20NG
vectorizer = TfidfVectorizer(max_features=5000)
ng20_data = vectorizer.fit_transform(ng20.data).toarray()
```

In [12]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

datasets = {
    'MNIST': (mnist_data, mnist_target),
    'Spambase': (spambase_data, spambase_target),
    '20NG': (ng20_data, ng20.target)
}

results = {}

for name, (data, target) in datasets.items():
    # Split data
    X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=

    # L2-reg Logistic Regression
    lr = LogisticRegression(penalty='l2', max_iter=5000).fit(X_train, y_train)
    lr_score = lr.score(X_test, y_test)

    # Decision Tree
    dt = DecisionTreeClassifier().fit(X_train, y_train)
    dt_score = dt.score(X_test, y_test)

    results[name] = {
        'Logistic Regression': (lr, lr_score),
        'Decision Tree': (dt, dt_score)
    }
```

In [14]:
```
import numpy as np

F = 30

for dataset, classifiers in results.items():
    for classifier, (model, score) in classifiers.items():
        print(f"Dataset: {dataset}, Classifier: {classifier}, Accuracy: {score:

        if classifier == 'Logistic Regression':
            coef = model.coef_
            # Check if coef is 2D (multiple classes) or 1D (binary classificati
            if len(coef.shape) == 2:
                top_features = np.argsort(np.abs(coef), axis=1)[:, -F:]
            else:
                top_features = np.argsort(np.abs(coef))[-F:]
                top_features = [top_features]  # Make it a 2D list for consiste
            print("Top Features:", top_features)

        elif classifier == 'Decision Tree':
            importances = model.feature_importances_
            top_features = np.argsort(importances)[-F:]
            print("Top Features:", top_features)
        print('-'*60)
```

```
Dataset: MNIST, Classifier: Logistic Regression, Accuracy: 0.9165
Top Features: [[117 329 323 395 406 377 229 690 322 434 605 444 145 374 282 6
29 309 710
  461 684 367 240 712 207 402 408 517 339 544 379]
 [240 428 381 691 481 591 174 342 313 359 383  89 473 429 374 201 277 520
  537 467 137 332 455 300 565 439 472 510 465 314]
 [542 247 465 354 165 325 509 360  97 320 715 674 219 607 705 359 370 258
  193 340 283 425 316 343 136 318 248 306 305 368]
 [130 358 394 300 186 710 510 657 572 397 249 267 695 565 520 136 128 566
  445 330 164 767 303 332 501 359  92 276 304 360]
 [357 322 444 542 624 584 186 306 323 546 321 563 376 593  67 570 320 535
  636 379 360 554 595  91 495 565 544 442  97 676]
 [389 416 522 565 289 248 182 442 737 654 385 244 427 283 268 257 472 386
  116  96 444  91 304 414  70 311 328 332 330 360]
 [640 387 360 543 203 631 606 503 301 313 267 593 270 621 323 340 567 269
  178 681 516 299 355 712 622 682 651 713 683 691]
 [539 564 639 349 752 328 459 418 569 120 266 132 594 595 404 432 278 610
  542 563 578 554 609 163 577 123 580 558 122 723]
 [ 75 278 652 722 397 203  74 341 579 228 360 377 552 580 619 554 379 480
  481 717 399 204 126 591 683 332 452 414 446 507]
 [599 621 359 199  97 255 647 361 156 606 328 133 555 227 648 620 572 444
  591 609 567 126 593 128 120 163 581 119 173 282]]
------------------------------------------------------------
Dataset: MNIST, Classifier: Decision Tree, Accuracy: 0.8731
Top Features: [598 490 515 372 380 658 355 404 297 596 484 657 656 405 296 29
0 206 353
 273 155 597 270 432 486 211 542 347 350 435 489]
------------------------------------------------------------
Dataset: Spambase, Classifier: Logistic Regression, Accuracy: 0.9305
Top Features: [[ 8 46  4 29  5 44 42  3 25 38 16 34 19 35 15 48 53 28 32 40 4
5 14 41 47
  24 43  6 22 52 26]]
------------------------------------------------------------
Dataset: Spambase, Classifier: Decision Tree, Accuracy: 0.9149
Top Features: [27 48 19 25 53  0  2 44  5  9 10 35 11 16 23 22 20 45 49 18  4
54 26 55
 15 56 24 51  6 52]
------------------------------------------------------------
Dataset: 20NG, Classifier: Logistic Regression, Accuracy: 0.6774
Top Features: [[4909  740 4446 4698 3966 4858 4487  700 4493  673 3616 3431 2
996 3917
  3133 2413 3581 2938 3740  693 2504 2939  735 2416 2417  542 3738  543
  2013  541]
 [4763 4756 2771 4159  320 3036 3413 2630  100 3957 1024 4755 4973 1988
  3407 4765 3525  365 4088 4540  418 1883 1093 1810 3441 1812 2271  142
  2270 2041]
 [1685  471 2741  713 4698 3133 1923  870 4750 1484 2241 1866 1505 2338
  4706 2964 1865 3206 3505 1812 1504 2761  585 3099 2977 4890  971 1810
  2863 4893]
 [2238 3505 2414 4397 1093  732 1484 2918 1851 2120 2836 4787 1442 2410
   812 3420 1954 1218 1167 2917 2931  162 2952 2244 3964 2930  870 3293
  1502  706]
 [1506  819 4690 2859 4763 2108 4798 1728  732 4104 1900 3479 2383 3133
  4213 3100 1502 3964 3443 2733 2264 2930 4105 2579 1518 3965 3599  909
   458 2729]
 [2303 1004  404 4479 2121 3172 1661 4966 4703 2155 4035 1024 4893 4326
   459 3791 4968 4698 4962 4881 4973 1444 4352 2905 4880 4964 4972 4028
  4892 2953]
 [3766 2689 3139 4815 4501 2930  767 1502 3116 1484  848 3203  841 2377
   899 1680 4001 1890 3479 4487    0 1108 3052  516 4000 4489 1872 4063
```

```
  3905 3137]
 [1924 1265  279 4861 1608 4725 3922 1507 1845 2364 4556 3479 4213 4631
  3673  575 3844 3133 2997 4801 2199  572 2063 4588 1296 3150 1876 1604
   881  869]
 [2655 4861 2364 1300 2122 2025 1184 4281 2679 1604 3673 2199 2565 3844
  4566 1921  916 1507 3826 3001 1473 2958 3828 2142  730 2957 3825  699
  1470  698]
 [3971 1000 3388 3357 2591 2169 4437 2854 2175 2455 3334  873 3360 3359
  4235 4190  612 4263 3884 2172  768 3358 3389 1947 1248 4980 2121 1946
  4436  624]
 [1397 3577 1387  794 3301 4895 1855 2419 2240 1925 3392 2012 1022 3388
  2590 3641 1252  789 3391 3389 2121 3386 1642 3854 3971 1947 3060 4436
  1946 2184]
 [4986 1242 1347 3985 4790 3336 2846 3570 3317 1639 3979 4623 1245 1006
  3875 1594 1287 3498  365 3326 2509 1358 3986  952 1243 2028 3098 2507
  1595 1007]
 [2706 3552  633 1255 2601 4092 1383 1107  403  798  563  975 4494  887
  4699 1187 3221 3950 4641 2055 3442 4902 3628 2959  952 3630  634 4793
  1576  974]
 [3946 2334 3342  737  965 4830  266 4982 2413 1558 3943 4129  893  723
  4394 2824 1867 3286 4610 1555 3133  855 2126 2827 3244 1408 1440 2979
  2826 1462]
 [3190 4823 3919 2813 3373 2551 3001 1292 3943  370  537 1948 2900 1935
  2713 3531 1848  548 2223 3918 4160 4193 1529 2570 3013 4130 2937 4082
  3189 4192]
 [4487 2151 3205 4624 2703 3807 4220 3084 4836 4748  740 1533 4871 3133
  2787 1750  693 2693 3962 2133  890 2783 4110 2451  962  960  963  961
   966 2013]
 [1087 2216 4854 1096 4505  521 1232  700 2397 1038 1235 1829 4838 1139
  1953 3096  632  642 2873 4806 2028  540 2572 4478 1828 1830 1776 4839
  2075 2074]
 [ 316 3410 4020 3249 2994 3297 4618  487 2423 4988 4636 2190 2298 3133
  4662 4856 2976  301 4633 3248 2995  476  498  499 2455 4634  475 2456
  2422 2421]
 [3318 3754 2216 4258 2626 1540  950 4491 4426 1511 2573 4043 2364 2126
  4254 2976 4044 2460 2837  813 4612 1512 2196 3469  632 1956 2198 3302
  1006 2028]
 [3740 2943 1249  439 3669 2121  668 2939 2938 4730 3192 1670 4515  265
  4652  950 4499  943 1776 3133 4487  963 3738 3114  693  961 2451 2538
  2504 2013]]
------------------------------------------------------------
Dataset: 20NG, Classifier: Decision Tree, Accuracy: 0.4174
Top Features: [ 966  458 4063  624 1595 2953 4988 2428 4512  498 1946 2041  4
09 2298
 2729 2074 3133 2413 4487 4192 4489 4556 2184 2421 1007  869  698 3905
 2013 4893]
------------------------------------------------------------
```

PROBLEM 2 : PCA library on MNIST

In [15]:
```python
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

def pca_and_classify(name, data, target, D):
    # Train-Test Split
    X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=

    # PCA with D features
    pca = PCA(n_components=D)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)

    classifiers = {
        'Logistic Regression': LogisticRegression(penalty='l2', max_iter=5000),
        'Decision Tree': DecisionTreeClassifier()
    }

    scores = {}

    for clf_name, clf in classifiers.items():
        clf.fit(X_train_pca, y_train)
        y_pred = clf.predict(X_test_pca)
        scores[clf_name] = accuracy_score(y_test, y_pred)

    return scores

# Problem 2 - A)
mnist_scores_5 = pca_and_classify('MNIST', mnist_data, mnist_target, 5)
mnist_scores_20 = pca_and_classify('MNIST', mnist_data, mnist_target, 20)
print(f"MNIST scores with D=5: {mnist_scores_5}")
print(f"MNIST scores with D=20: {mnist_scores_20}")

# Problem 2 - B)
def find_optimal_D_for_spambase():
    best_D = None
    original_scores = results['Spambase']

    for D in range(1, spambase_data.shape[1] + 1):  # Iterate from 1 to number o
        pca_scores = pca_and_classify('Spambase', spambase_data, spambase_target

        if all(abs(pca_scores[clf] - original_scores[clf][1]) < 0.01 for clf in
            best_D = D
            break

    return best_D, pca_scores

best_D, spambase_scores = find_optimal_D_for_spambase()
print(f"Optimal D for Spambase: {best_D}, with scores: {spambase_scores}")
```

```
MNIST scores with D=5: {'Logistic Regression': 0.6786285714285715, 'Decision
Tree': 0.6704}
MNIST scores with D=20: {'Logistic Regression': 0.8684571428571428, 'Decision
Tree': 0.8443428571428572}
Optimal D for Spambase: None, with scores: {'Logistic Regression': 0.93049522
15464813, 'Decision Tree': 0.8592528236316247}
```

PROBLEM 3 : Implement PCA on MNIST

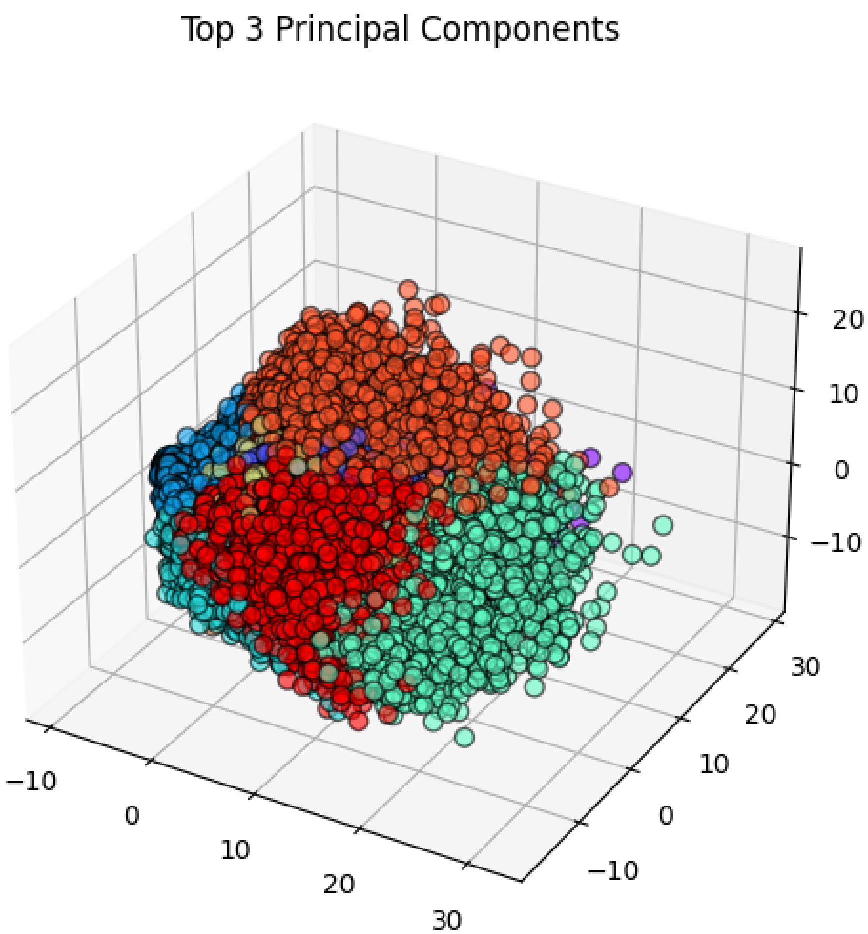
In [17]:
```python
import numpy as np

class CustomPCA:
    def __init__(self, n_components):
        self.n_components = n_components
        self.eigenvectors = None

    def fit_transform(self, X):
        # 1. Standardize the dataset
        X_standardized = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

        # 2. Compute the covariance matrix
        covariance_matrix = np.cov(X_standardized, rowvar=False)

        # 3. Compute the eigenvectors and eigenvalues
        eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)

        # 4. Sort eigenvalues and eigenvectors in descending order
        sorted_indices = np.argsort(eigenvalues)[::-1]
        self.eigenvectors = eigenvectors[:, sorted_indices[:self.n_components]]

        # 5. Project the data onto the lower-dimensional subspace
        return np.dot(X_standardized, self.eigenvectors)

# Testing our CustomPCA on MNIST data for D=5 and D=20
D_values = [5, 20]
mnist_pca_results = {}

for D in D_values:
    pca = CustomPCA(n_components=D)
    mnist_data_pca = pca.fit_transform(mnist_data)

    # Handle NaNs
    mnist_data_pca = np.nan_to_num(mnist_data_pca)

    # Reuse the pca_and_classify function from previous code
    scores = pca_and_classify('MNIST', mnist_data_pca, mnist_target, D)
    mnist_pca_results[D] = scores

print("MNIST results using custom PCA:")
for D, scores in mnist_pca_results.items():
    print(f"For D={D}, scores: {scores}")
```

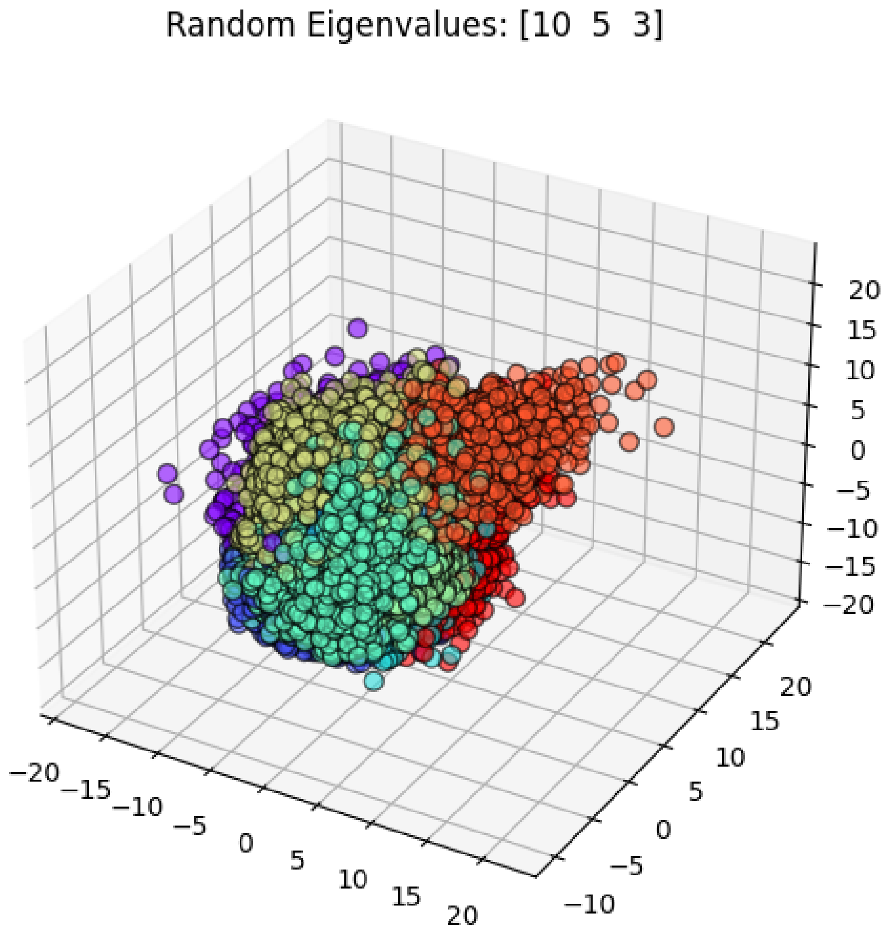
```
/var/folders/2d/kjz0bk3s5nj4p4v10t35f8f40000gn/T/ipykernel_3969/4093399788.p
y:10: RuntimeWarning: invalid value encountered in divide
  X_standardized = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
/usr/local/lib/python3.10/site-packages/sklearn/decomposition/_pca.py:545: Ru
ntimeWarning: invalid value encountered in divide
  explained_variance_ratio_ = explained_variance_ / total_var
/var/folders/2d/kjz0bk3s5nj4p4v10t35f8f40000gn/T/ipykernel_3969/4093399788.p
y:10: RuntimeWarning: invalid value encountered in divide
  X_standardized = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
/usr/local/lib/python3.10/site-packages/sklearn/decomposition/_pca.py:545: Ru
ntimeWarning: invalid value encountered in divide
  explained_variance_ratio_ = explained_variance_ / total_var

MNIST results using custom PCA:
For D=5, scores: {'Logistic Regression': 0.11297142857142857, 'Decision Tre
e': 0.11297142857142857}
For D=20, scores: {'Logistic Regression': 0.11297142857142857, 'Decision Tre
e': 0.11297142857142857}
```

PROBLEM 4 : PCA for cluster visualization

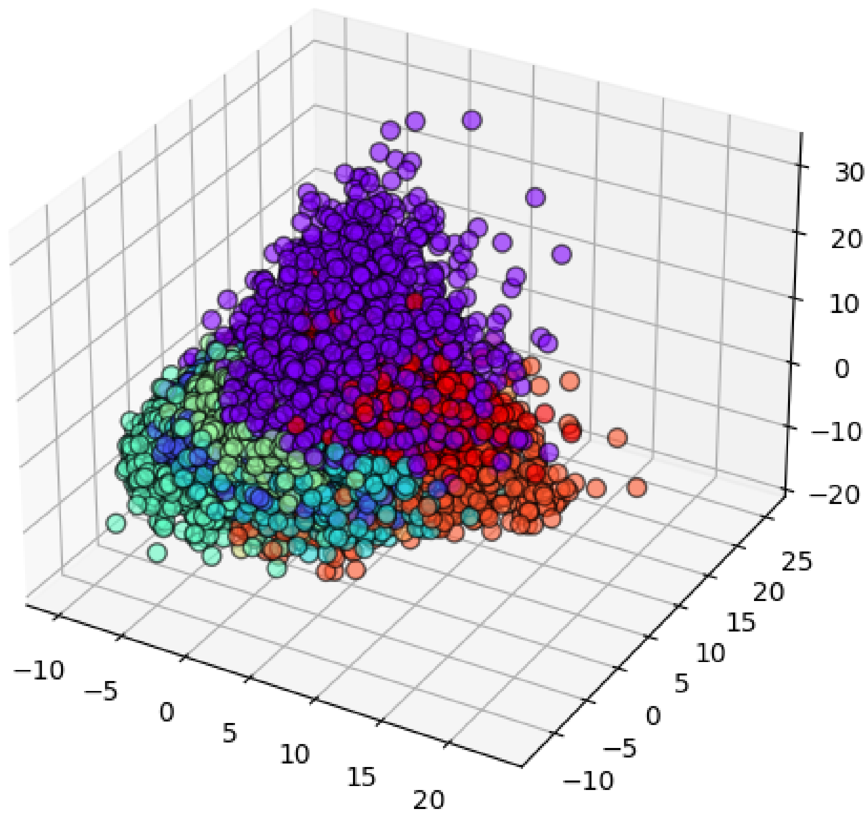
In [18]:
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

# 1. Run KMeans on MNIST data.
kmeans = KMeans(n_clusters=10)
clusters = kmeans.fit_predict(mnist_data)

# 2. Perform PCA on the same data.
pca = PCA(n_components=20) # We'll extract the top 20 components for future use
mnist_pca = pca.fit_transform(mnist_data)

# Helper function to plot 3D
def plot_3d(data, clusters, indices, title):
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')

    scatter = ax.scatter(data[:, indices[0]], data[:, indices[1]], data[:, indic
                         c=clusters, cmap='rainbow', marker='o', edgecolor='k',
    ax.set_title(title)
    plt.show()

# 3. Visualize the data in 3D using PCA with the top 3 eigenvalues.
plot_3d(mnist_pca, clusters, [0, 1, 2], "Top 3 Principal Components")

# 4. Randomly select 3 eigenvalues from the top 20 and visualize them in 3D.
for _ in range(3): # Repeating 3 times
    random_indices = np.random.choice(range(20), 3, replace=False)
    plot_3d(mnist_pca, clusters, random_indices, f"Random Eigenvalues: {random_
```

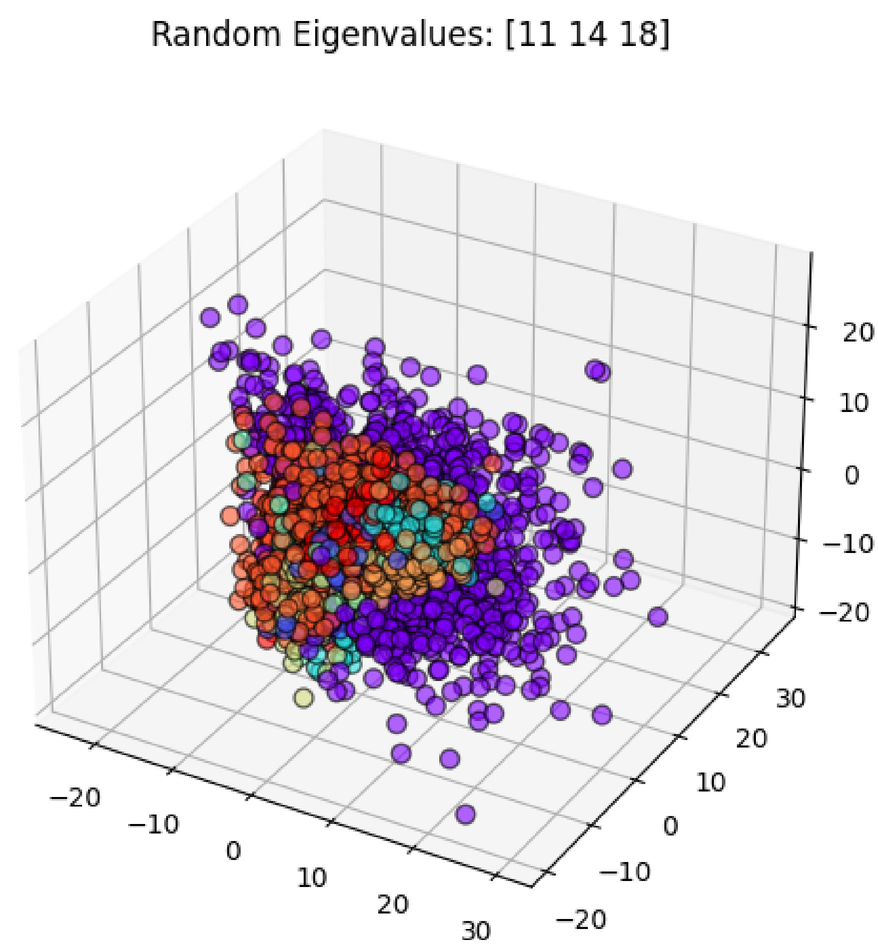
```
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: Futur
eWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4.
Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```


Top 3 Principal Components
20
10
0
−10
30
20
10
0
−10
−10
0
10
20
30


Random Eigenvalues: [10 5 3]
20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20
20
15
10
5
0
−5
−10

Random Eigenvalues: [5 7 6]




Random Eigenvalues: [11 14 18]
20
10
0
−10
−20
30
20
10
0
−10
−20
−20
−10
0
10
20
30

PROBLEM 5 : Implement Kernel PCA for linear regression

In [19]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import KernelPCA

# 1. Load the datasets
two_spirals = pd.read_csv('twoSpirals.txt', sep='\t', header=None)
three_circles = pd.read_csv('threecircles.txt', sep=',', header=None)

datasets = {
    "twoSpirals": two_spirals,
    "threeCircles": three_circles
}

for name, data in datasets.items():
    X = data.iloc[:, :-1].values
    y = data.iloc[:, -1].values

    # 2. Train a Linear Regression on the original data
    reg = LinearRegression().fit(X, y)
    predictions = reg.predict(X)
    mse = mean_squared_error(y, predictions)
    print(f"Original MSE for {name}: {mse:.3f}")

    # 3. Implement KernelPCA with Gaussian Kernel
    X2 = np.sum(X*X, axis=1).reshape(-1, 1)
    DIST_euclid = X2 + X2.T - 2 * np.dot(X, X.T)
    sigma = 3
    K = np.exp(-DIST_euclid/sigma)
    N = K.shape[0]
    U = np.ones((N, N)) / N
    Kn = K - np.dot(U, K) - np.dot(K, U) + np.dot(U, np.dot(K, U))

    eigenvalues, eigenvectors = np.linalg.eigh(Kn)
    sorted_indices = np.argsort(eigenvalues)[::-1]
    eigenvectors = eigenvectors[:, sorted_indices]
    eigenvalues = eigenvalues[sorted_indices]

    # 4. Retrain Linear regression on the transformed D-dim data
    for D in [3, 20, 100]:
        X_transformed = np.dot(Kn, eigenvectors[:, :D])
        reg_transformed = LinearRegression().fit(X_transformed, y)
        predictions_transformed = reg_transformed.predict(X_transformed)
        mse_transformed = mean_squared_error(y, predictions_transformed)
        print(f"MSE for {name} with D={D}: {mse_transformed:.3f}")
```

```
Original MSE for twoSpirals: 0.839
MSE for twoSpirals with D=3: 0.838
MSE for twoSpirals with D=20: 0.181
MSE for twoSpirals with D=100: 0.008
Original MSE for threeCircles: 0.666
MSE for threeCircles with D=3: 0.030
MSE for threeCircles with D=20: 0.005
MSE for threeCircles with D=100: 0.000
```