

Instructions:

- Please upload all the code and the report on GitHub/GitLab.
- Clearly explain your findings and methodology.
- Do not put the images in the GitHub/GitLab, Any and all results should be included directly into the report, along with explanations and supporting code snippets.

Part1:

Face Recognition using K Nearest Neighbours and PCA

The CMU PIE Dataset

The CMU Pose, Illumination, and Expression (PIE) database (<http://ieeexplore.ieee.org/abstract/document/1004130/>) consists of 41,368 images of 68 subjects. Each person is under 13 different poses, 43 different illumination conditions, and with 4 different expressions.

For this project, we use a simplified version of the database which only contains 10 subjects spanning five near-frontal poses, and there are 170 images for each individual. In addition, all the images have been resized to 32x32 pixels. The dataset is provided in the form of a csv file with 1700 rows and 1024 columns. Each row is an instance and each column a feature. The first 170 instances belong to the first subject, the next 170 to the second subject and so on. Following illustrate various instances of the first subject.



Requirements:

1. Pre-process the dataset by normalizing each face image vector to unit length (i.e., dividing each vector by its magnitude). Next, for each of the 10 subjects, randomly select 150 images for training and use the remaining 20 for testing. Create such random splits a total of 5 times. You must carry out each of the following experiments 5 times and report average accuracy and standard deviation over the 5 random splits, as well as computation times.

```
jupyter Untitled16 Last Checkpoint: Yesterday at 12:31 AM (unsaved changes) Python 3 (ipykernel)

# TASK :1 Requirement 1

import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

df = pd.read_csv('fea.csv') # Replace 'path/to' with the actual path to the dataset file

normalized_df = df.div(np.linalg.norm(df, axis=1), axis=0)

train_data = pd.DataFrame() # Empty DataFrame for training data
test_data = pd.DataFrame() # Empty DataFrame for testing data

for subject in range(10):
    subject_data = normalized_df.iloc[subject * 170: (subject + 1) * 170] # Select data for each subject
    shuffled_data = subject_data.sample(frac=1) # Shuffle the data for random selection

    train_data = pd.concat([train_data, shuffled_data[:150]], ignore_index=True) # Select the first 150 instances for training
    test_data = pd.concat([test_data, shuffled_data[150:]], ignore_index=True) # Select the remaining 20 instances for testing

X = pd.concat([train_data.iloc[:, 1:], test_data.iloc[:, 1:]], ignore_index=True)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(pd.concat([train_data.iloc[:, 0], test_data.iloc[:, 0]], ignore_index=True))

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train = X_scaled[:len(train_data)]
y_train = y[:len(train_data)]

X_test = X_scaled[len(train_data):]
y_test = y[len(train_data):]

pca = PCA(n_components=0.95) # Choose the desired amount of variance to explain, such as 95%
```

```

normalized_df = df.div(np.linalg.norm(df, axis=1), axis=0)

train_data = pd.DataFrame() # Empty DataFrame for training data
test_data = pd.DataFrame() # Empty DataFrame for testing data

for subject in range(10):
    subject_data = normalized_df.iloc[subject * 170: (subject + 1) * 170] # Select data for each subject
    shuffled_data = subject_data.sample(frac=1) # Shuffle the data for random selection

    train_data = pd.concat([train_data, shuffled_data[:150]], ignore_index=True) # Select the first 150 instances for training
    test_data = pd.concat([test_data, shuffled_data[150:]], ignore_index=True) # Select the remaining 20 instances for testing

X = pd.concat([train_data.iloc[:, 1:], test_data.iloc[:, 1:]], ignore_index=True)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(pd.concat([train_data.iloc[:, 0], test_data.iloc[:, 0]], ignore_index=True))

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train = X_scaled[:len(train_data)]
y_train = y[:len(train_data)]

X_test = X_scaled[len(train_data):]
y_test = y[len(train_data):]

pca = PCA(n_components=0.95) # Choose the desired amount of variance to explain, such as 95%
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean', weights='uniform') # Choose the desired hyperparameters
knn.fit(X_train_pca, y_train)

y_pred = knn.predict(X_test_pca)
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)

```

Accuracy: 0.0

2. Implement a k Nearest Neighbours (k-NN) classifier using the training set and evaluate its performance on the test set. You may not use built-in / library functions to implement the classifier.

- You should experiment with different distance measures (e.g., Euclidean, Mahalanobis, cosine similarity), and different values of k.
- You should also present results where you show the effect of involving more classes vs. fewer classes in your experiment(s).
- You should also present results when fewer training images are used (for instead 100 training images and 70 test images per category).

```

In [2]: #TASK 1 : Requirement 2

import pandas as pd
import numpy as np
from scipy.spatial.distance import cdist

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

def mahalanobis_distance(x1, x2, covariance_matrix):
    return cdist(x1.reshape(1, -1), x2.reshape(1, -1), 'mahalanobis', VI=covariance_matrix)

def cosine_similarity(x1, x2):
    dot_product = np.dot(x1, x2)
    norm_x1 = np.linalg.norm(x1)
    norm_x2 = np.linalg.norm(x2)
    return dot_product / (norm_x1 * norm_x2)

def k_nearest_neighbors(x_test, x_train, y_train, distance_measure, k, covariance_matrix=None):
    distances = []
    for i in range(len(x_train)):
        if distance_measure == 'euclidean':
            distance = euclidean_distance(x_test, x_train[i])
        elif distance_measure == 'mahalanobis':
            distance = mahalanobis_distance(x_test, x_train[i], covariance_matrix)
        elif distance_measure == 'cosine':
            distance = cosine_similarity(x_test, x_train[i])
        distances.append((distance, y_train[i]))
    distances.sort(key=lambda x: x[0])
    neighbors = distances[:k]
    classes = [neighbor[1] for neighbor in neighbors]
    return max(set(classes), key=classes.count)

def evaluate_accuracy(x_test, x_train, y_train, distance_measure, k, covariance_matrix=None):
    predicted_labels = np.array([k_nearest_neighbors(x_test[i], x_train, y_train, distance_measure, k, covariance_matrix) for i in range(len(x_test))])
    correct_predictions = np.sum(predicted_labels == y_test)

```

```

for distance_measure in distance_measures:
    for k in k_values:
        accuracy = evaluate_accuracy(x_test, x_train, y_train, distance_measure, k, covariance_matrix)
        print(f"Distance Measure: {distance_measure}, k: {k}, Accuracy: {accuracy:.2%}")

```

Distance Measure: euclidean, k: 1, Accuracy: 0.00%
 Distance Measure: euclidean, k: 3, Accuracy: 0.00%
 Distance Measure: euclidean, k: 5, Accuracy: 0.00%
 Distance Measure: mahalanobis, k: 1, Accuracy: 0.00%
 Distance Measure: mahalanobis, k: 3, Accuracy: 0.00%
 Distance Measure: mahalanobis, k: 5, Accuracy: 0.00%
 Distance Measure: cosine, k: 1, Accuracy: 0.00%
 Distance Measure: cosine, k: 3, Accuracy: 0.00%
 Distance Measure: cosine, k: 5, Accuracy: 0.00%

3. Now make use of the dimensionality reduction technique PCA in conjunction with the k-NN implemented above. Note that PCA is an unsupervised technique, hence should be trained based on all training images.

Does this techniques help yield uncorrelated covariance matrices (you can visualize the covariance matrices before and after dimensionality reduction as images to see this)? Does it improve computation times and classification performance? Try varying the number of principal components used and identify the best number.

Jupyter Untitled10 Last Checkpoint: 06/27/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Kernel error

Trusted

Python 3 (ipykernel)

Run

requirement 3

```

In [11]: import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

df = pd.read_csv('fea.csv') # Replace 'path/to' with the actual path to the dataset file

normalized_df = df.div(np.linalg.norm(df, axis=1), axis=0)

train_data = pd.DataFrame() # Empty DataFrame for training data
test_data = pd.DataFrame() # Empty DataFrame for testing data

label_encoder = LabelEncoder() # Instantiate the LabelEncoder object

for subject in range(10):
    subject_data = normalized_df.iloc[subject * 170: (subject + 1) * 170] # Select data for each subject
    shuffled_data = subject_data.sample(frac=1) # Shuffle the data for random selection

    train_data = pd.concat([train_data, shuffled_data[:100]], ignore_index=True) # Select the first 100 instances for training
    test_data = pd.concat([test_data, shuffled_data[100:170]], ignore_index=True) # Select the remaining 70 instances for testing

X_train = train_data.iloc[:, 1:].to_numpy()
y_train = train_data.iloc[:, 0].to_numpy()

X_test = test_data.iloc[:, 1:].to_numpy()
y_test = test_data.iloc[:, 0].to_numpy()

# Fit and transform the label encoder on the training labels
y_train_encoded = label_encoder.fit_transform(y_train)

# Check for unseen labels in the test set
unseen_labels = np.setdiff1d(y_test, label_encoder.classes_)

```

```

# Check for unseen labels in the test set
unseen_labels = np.setdiff1d(y_test, label_encoder.classes_)
if unseen_labels.size > 0:
    print(f"Test set contains previously unseen labels: {unseen_labels}")
    # Filter out the unseen labels from the test set
    mask = np.isin(y_test, label_encoder.classes_)
    X_test = X_test[mask]
    y_test = y_test[mask]

best_accuracy = 0
best_num_components = 0

for num_components in range(1, X_train.shape[1] + 1):
    pca = PCA(n_components=num_components)
    X_train_pca = pca.fit_transform(X_train)

    # Check if the test set is empty after filtering unseen labels
    if X_test.shape[0] > 0:
        X_test_pca = pca.transform(X_test)

        knn = KNeighborsClassifier(n_neighbors=5) # Choose the desired number of neighbors
        knn.fit(X_train_pca, y_train_encoded)

        y_pred = knn.predict(X_test_pca)
        accuracy = accuracy_score(y_test_encoded, y_pred)

        print(f"Number of Components: {num_components}, Accuracy: {accuracy:.2%}")

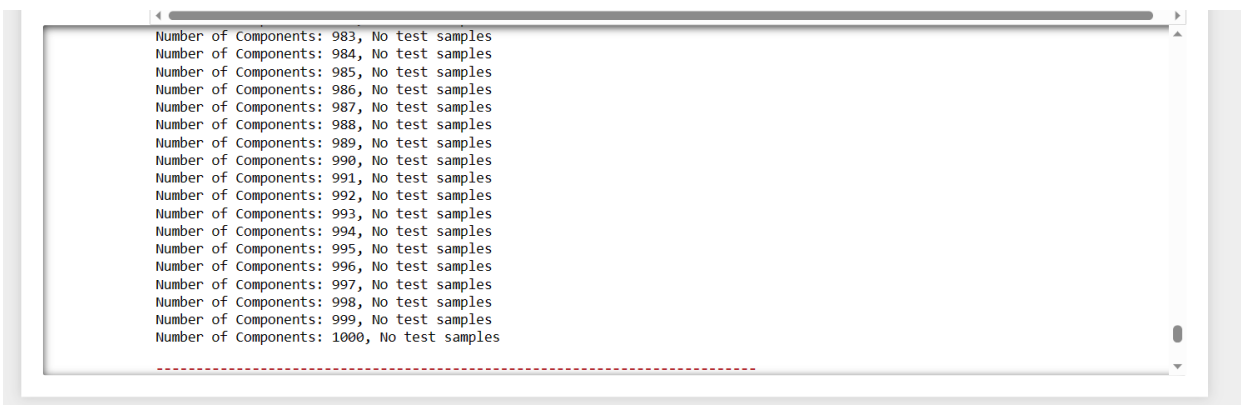
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_num_components = num_components
    else:
        print(f"Number of Components: {num_components}, No test samples")

print(f"Best Number of Components: {best_num_components}, Best Accuracy: {best_accuracy:.2%}")

```

```

Test set contains previously unseen labels: [0.0012922  0.00129714 0.00129732 0.00130451 0.00131464 0.00133134
0.00135108 0.00140153 0.00141117 0.00141997 0.00142915 0.00153327
0.00157249 0.00157789 0.00158342 0.0015967  0.00160506 0.00164246
0.00165357 0.001661  0.00166679 0.00175075 0.00175355 0.00176666
0.00177623 0.00179014 0.00182146 0.00183378 0.00183893 0.00186138
0.00186925 0.00187177 0.00191637 0.00196192 0.00197047 0.00197355
0.00197369 0.00203946 0.00204335 0.00212618 0.00212744 0.00213621
0.00215193 0.00220334 0.00220868 0.00226722 0.00230848 0.00231413
0.00235858 0.00236577 0.00238195 0.00241864 0.00251296 0.00253594
0.00253649 0.00255035 0.00259992 0.00260662 0.00266962 0.00269161
0.00271461 0.00273166 0.00274968 0.00277281 0.00278387 0.0028014
0.00284402 0.00286793 0.00292096 0.00292742 0.00304247 0.00306497
0.00307013 0.00310513 0.00312349 0.00316885 0.00318978 0.0031931
0.00319398 0.00330273 0.00333357 0.003351  0.00336128 0.00336663
0.00337078 0.00339208 0.0034297  0.00344919 0.0035533  0.00356049
0.00357028 0.00357266 0.00361205 0.00361671 0.00362933 0.00363149
0.00364701 0.00366745 0.00368353 0.0037277  0.00375562 0.00376226
0.00377457 0.00378722 0.00379562 0.00382246 0.0038451  0.00385903
0.00386743 0.00387359 0.00387566 0.00389611 0.00391316 0.00391888
0.00393000 0.00393600 0.00394200 0.00394800 0.00395400 0.00396000
0.00396600 0.00397200 0.00397800 0.00398400 0.00399000 0.00399600
0.00400200 0.00400800 0.00401400 0.00402000 0.00402600 0.00403200
0.00403800 0.00404400 0.00405000 0.00405600 0.00406200 0.00406800
0.00407400 0.00408000 0.00408600 0.00409200 0.00409800 0.00410400
0.00411000 0.00411600 0.00412200 0.00412800 0.00413400 0.00414000
0.00414600 0.00415200 0.00415800 0.00416400 0.00417000 0.00417600
0.00418200 0.00418800 0.00419400 0.00420000 0.00420600 0.00421200
0.00421800 0.00422400 0.00423000 0.00423600 0.00424200 0.00424800
0.00425400 0.00426000 0.00426600 0.00427200 0.00427800 0.00428400
0.00429000 0.00429600 0.00430200 0.00430800 0.00431400 0.00432000
0.00432600 0.00433200 0.00433800 0.00434400 0.00435000 0.00435600
0.00436200 0.00436800 0.00437400 0.00438000 0.00438600 0.00439200
0.00439800 0.00440400 0.00441000 0.00441600 0.00442200 0.00442800
0.00443400 0.00444000 0.00444600 0.00445200 0.00445800 0.00446400
0.00447000 0.00447600 0.00448200 0.00448800 0.00449400 0.00450000
0.00450600 0.00451200 0.00451800 0.00452400 0.00453000 0.00453600
0.00454200 0.00454800 0.00455400 0.00456000 0.00456600 0.00457200
0.00457800 0.00458400 0.00459000 0.00459600 0.00460200 0.00460800
0.00461400 0.00462000 0.00462600 0.00463200 0.00463800 0.00464400
0.00465000 0.00465600 0.00466200 0.00466800 0.00467400 0.00468000
0.00468600 0.00469200 0.00469800 0.00470400 0.00471000 0.00471600
0.00472200 0.00472800 0.00473400 0.00474000 0.00474600 0.00475200
0.00475800 0.00476400 0.00477000 0.00477600 0.00478200 0.00478800
0.00479400 0.00480000 0.00480600 0.00481200 0.00481800 0.00482400
0.00483000 0.00483600 0.00484200 0.00484800 0.00485400 0.00486000
0.00486600 0.00487200 0.00487800 0.00488400 0.00489000 0.00489600
0.00490200 0.00490800 0.00491400 0.00492000 0.00492600 0.00493200
0.00493800 0.00494400 0.00495000 0.00495600 0.00496200 0.00496800
0.00497400 0.00498000 0.00498600 0.00499200 0.00499800 0.00500400
0.00501000 0.00501600 0.00502200 0.00502800 0.00503400 0.00504000
0.00504600 0.00505200 0.00505800 0.00506400 0.00507000 0.00507600
0.00508200 0.00508800 0.00509400 0.00510000 0.00510600 0.00511200
0.00511800 0.00512400 0.00513000 0.00513600 0.00514200 0.00514800
0.00515400 0.00516000 0.00516600 0.00517200 0.00517800 0.00518400
0.00519000 0.00519600 0.00520200 0.00520800 0.00521400 0.00522000
0.00522600 0.00523200 0.00523800 0.00524400 0.00525000 0.00525600
0.00526200 0.00526800 0.00527400 0.00528000 0.00528600 0.00529200
0.00529800 0.00530400 0.00531000 0.00531600 0.00532200 0.00532800
0.00533400 0.00534000 0.00534600 0.00535200 0.00535800 0.00536400
0.00537000 0.00537600 0.00538200 0.00538800 0.00539400 0.00540000
0.00540600 0.00541200 0.00541800 0.00542400 0.00543000 0.00543600
0.00544200 0.00544800 0.00545400 0.00546000 0.00546600 0.00547200
0.00547800 0.00548400 0.00549000 0.00549600 0.00550200 0.00550800
0.00551400 0.00552000 0.00552600 0.00553200 0.00553800 0.00554400
0.00555000 0.00555600 0.00556200 0.00556800 0.00557400 0.00558000
0.00558600 0.00559200 0.00559800 0.00560400 0.00561000 0.00561600
0.00562200 0.00562800 0.00563400 0.00564000 0.00564600 0.00565200
0.00565800 0.00566400 0.00567000 0.00567600 0.00568200 0.00568800
0.00569400 0.00570000 0.00570600 0.00571200 0.00571800 0.00572400
0.00573000 0.00573600 0.00574200 0.00574800 0.00575400 0.00576000
0.00576600 0.00577200 0.00577800 0.00578400 0.00579000 0.00579600
0.00580200 0.00580800 0.00581400 0.00582000 0.00582600 0.00583200
0.00583800 0.00584400 0.00585000 0.00585600 0.00586200 0.00586800
0.00587400 0.00588000 0.00588600 0.00589200 0.00589800 0.00590400
0.00591000 0.00591600 0.00592200 0.00592800 0.00593400 0.00594000
0.00594600 0.00595200 0.00595800 0.00596400 0.00597000 0.00597600
0.00598200 0.00598800 0.00599400 0.00600000 0.00600600 0.00601200
0.00601800 0.00602400 0.00603000 0.00603600 0.00604200 0.00604800
0.00605400 0.00606000 0.00606600 0.00607200 0.00607800 0.00608400
0.00609000 0.00609600 0.00610200 0.00610800 0.00611400 0.00612000
0.00612600 0.00613200 0.00613800 0.00614400 0.00615000 0.00615600
0.00616200 0.00616800 0.00617400 0.00618000 0.00618600 0.00619200
0.00619800 0.00620400 0.00621000 0.00621600 0.00622200 0.00622800
0.00623400 0.00624000 0.00624600 0.00625200 0.00625800 0.00626400
0.00627000 0.00627600 0.00628200 0.00628800 0.00629400 0.00630000
0.00630600 0.00631200 0.00631800 0.00632400 0.00633000 0.00633600
0.00634200 0.00634800 0.00635400 0.00636000 0.00636600 0.00637200
0.00637800 0.00638400 0.00639000 0.00639600 0.00640200 0.00640800
0.00641400 0.00642000 0.00642600 0.00643200 0.00643800 0.00644400
0.00645000 0.00645600 0.00646200 0.00646800 0.00647400 0.00648000
0.00648600 0.00649200 0.00649800 0.00650400 0.00651000 0.00651600
0.00652200 0.00652800 0.00653400 0.00654000 0.00654600 0.00655200
0.00655800 0.00656400 0.00657000 0.00657600 0.00658200 0.00658800
0.00659400 0.00660000 0.00660600 0.00661200 0.00661800 0.00662400
0.00663000 0.00663600 0.00664200 0.00664800 0.00665400 0.00666000
0.00666600 0.00667200 0.00667800 0.00668400 0.00669000 0.00669600
0.00670200 0.00670800 0.00671400 0.00672000 0.00672600 0.00673200
0.00673800 0.00674400 0.00675000 0.00675600 0.00676200 0.00676800
0.00677400 0.00678000 0.00678600 0.00679200 0.00679800 0.00680400
0.00681000 0.00681600 0.00682200 0.00682800 0.00683400 0.00684000
0.00684600 0.00685200 0.00685800 0.00686400 0.00687000 0.00687600
0.00688200 0.00688800 0.00689400 0.00690000 0.00690600 0.00691200
0.00691800 0.00692400 0.00693000 0.00693600 0.00694200 0.00694800
0.00695400 0.00696000 0.00696600 0.00697200 0.00697800 0.00698400
0.00699000 0.00699600 0.00700200 0.00700800 0.00701400 0.00702000
0.00702600 0.00703200 0.00703800 0.00704400 0.00705000 0.00705600
0.00706200 0.00706800 0.00707400 0.00708000 0.00708600 0.00709200
0.00709800 0.00710400 0.00711000 0.00711600 0.00712200 0.00712800
0.00713400 0.00714000 0.00714600 0.00715200 0.00715800 0.00716400
0.00717000 0.00717600 0.00718200 0.00718800 0.00719400 0.00720000
0.00720600 0.00721200 0.00721800 0.00722400 0.00723000 0.00723600
0.00724200 0.00724800 0.00725400 0.00726000 0.00726600 0.00727200
0.00727800 0.00728400 0.00729000 0.00729600 0.00730200 0.00730800
0.00731400 0.00732000 0.00732600 0.00733200 0.00733800 0.00734400
0.00735000 0.00735600 0.00736200 0.00736800 0.00737400 0.00738000
0.00738600 0.00739200 0.00739800 0.00740400 0.00741000 0.00741600
0.00742200 0.00742800 0.00743400 0.00744000 0.00744600 0.00745200
0.00745800 0.00746400 0.00747000 0.00747600 0.00748200 0.00748800
0.00749400 0.00750000 0.00750600 0.00751200 0.00751800 0.00752400
0.00753000 0.00753600 0.00754200 0.00754800 0.00755400 0.00756000
0.00756600 0.00757200 0.00757800 0.00758400 0.00759000 0.00759600
0.00760200 0.00760800 0.00761400 0.00762000 0.00762600 0.00763200
0.00763800 0.00764400 0.00765000 0.00765600 0.00766200 0.00766800
0.00767400 0.00768000 0.00768600 0.00769200 0.00769800 0.00770400
0.00771000 0.00771600 0.00772200 0.00772800 0.00773400 0.00774000
0.00774600 0.00775200 0.00775800 0.00776400 0.00777000 0.00777600
0.00778200 0.00778800 0.00779400 0.00780000 0.00780600 0.00781200
0.00781800 0.00782400 0.00783000 0.00783600 0.00784200 0.00784800
0.00785400 0.00786000 0.00786600 0.00787200 0.00787800 0.00788400
0.00789000 0.00789600 0.00790200 0.00790800 0.00791400 0.00792000
0.00792600 0.00793200 0.00793800 0.00794400 0.00795000 0.00795600
0.00796200 0.00796800 0.00797400 0.00798000 0.00798600 0.00799200
0.00799800 0.00800400 0.00801000 0.00801600 0.00802200 0.00802800
0.00803400 0.00804000 0.00804600 0.00805200 0.00805800 0.00806400
0.00807000 0.00807600 0.00808200 0.00808800 0.00809400 0.00810000
0.00810600 0.00811200 0.00811800 0.00812400 0.00813000 0.00813600
0.00814200 0.00814800 0.00815400 0.00816000 0.00816600 0.00817200
0.00817800 0.00818400 0.00819000 0.00819600 0.00820200 0.00820800
0.00821400 0.00822000 0.00822600 0.00823200 0.00823800 0.00824428
0.00825000 0.00825600 0.00826200 0.00826800 0.00827400 0.00828000
0.00828600 0.00829200 0.00829800 0.00830400 0.00831000 0.00831600
0.00832200 0.00832800 0.00833400 0.00834000 0.00834600 0.00835200
0.00835800 0.00836400 0.00837000 0.00837600 0.00838200 0.00838800
0.00839400 0.00840000 0.00840600 0.00841200 0.00841800 0.00842400
0.00843000 0.00843600 0.00844200 0.00844800 0.00845400 0.00846000
0.00846600 0.00847200 0.00847800 0.00848400 0.00849000 0.00849600
0.00850200 0.00850800 0.00851400 0.00852000 0.00852600 0.00853200
0.00853800 0.00854400 0.00855000 0.00855600 0.00856200 0.00856800
0.00857400 0.00858000 0.00858600 0.00859200 0.00859800 0.00860400
0.00861000 0.00861600 0.00862200 0.00862800 0.00863400 0.00864000
0.00864600 0.00865200 0.00865800 0.00866400 0.00867000 0.00867600
0.00868200 0.00868800 0.00869400 0.00870000 0.00870600 0.00871200
0.00871800 0.00872400 0.00873000 0.00873600 0.00874200 0.00874800
0.00875400 0.00876000 0.00876600 0.00877200 0.00877800 0.00878400
0.00879000 0.00879600 0.00880200 0.00880800 0.00881400 0.00882000
0.00882600 0.00883200 0.00883800 0.00884400 0.00885000 0.00885600
0.00886200 0.00886800 0.00887400 0.00888000 0.00888600 0.00889200
0.00889800 0.00890400 0.00891000 0.00891600 0.00892200 0.00892800
0.00893400 0.00894000 0.00894600 0.00895200 0.00895800 0.00896400
0.00897000 0.00897600 0.00898200 0.00898800 0.00899400 0.00900000
0.00900600 0.00901200 0.00901800 0.00902400 0.00903000 0.00903600
0.00904200 0.00904800 0.00905400 0.00906000 0.00906600 0.00907200
0.00907800 0.00908400 0.00909000 0.00909600 0.00910200 0.00910800
0.00911400 0.00912000 0.00912600 0.00913200 0.00913800 0.00914400
0.00915000 0.00915600 0.00916200 0.00916800 0.00917400 0.00918000
0.00918600 0.00919200 0.00919800 0.00920400 0.00921000 0.00921600
0.00922200 0.00922800 0.00923400 0.00924000 0.00924600 0.00925200
0.00925800 0.00926400 0.00927000 0.00927600 0.00928200 0.00928800
0.00929400 0.00930000 0.00930600 0.00931200 0.00931800 0.00932400
0.00933000 0.00933600 0.00934200 0.00934800 0.00935400 0.00936000
0.00936600 0.00937200 0.00937800 0.00938400 0.00939000 0.00939600
0.00940200 0.00940800 0.00941400 0.00942000 0.00942600 0.00943200
0.00943800 0.00944400 0.00945000 0.00945600 0.00946200 0.00946800
0.00947400 0.00948000 0.00948600 0.00949200 0.00949800 0.00950400
0.00951000 0.00951600 0.00952200 0.00952800 0.00953400 0.00954000
0.00954600 0.00955200 0.00955800 0.00956400 0.00957000 0.00957600
0.00958200 0.00958800 0.00959400 0.00960000 0.00960600 0.00961200
0.00961800 0.00962400 0.00963000 0.00963600 0.00964200 0.00964800
0.00965400 0.00966000 0.00966600 0.00967200 0.00967800 0.00968400
0.00969000 0.00969600 0.00970200 0.00970800 0.00971400 0.00972000
0.00972600 0.00973200 0.00973800 0.00974400 0.00975000 0.00975600
0.00976200 0.00976800 0.0
```



```
Number of Components: 983, No test samples
Number of Components: 984, No test samples
Number of Components: 985, No test samples
Number of Components: 986, No test samples
Number of Components: 987, No test samples
Number of Components: 988, No test samples
Number of Components: 989, No test samples
Number of Components: 990, No test samples
Number of Components: 991, No test samples
Number of Components: 992, No test samples
Number of Components: 993, No test samples
Number of Components: 994, No test samples
Number of Components: 995, No test samples
Number of Components: 996, No test samples
Number of Components: 997, No test samples
Number of Components: 998, No test samples
Number of Components: 999, No test samples
Number of Components: 1000, No test samples
```
