Homework 3

CSCE 633-600

Ada Huang

March 20, 2023

In [ ]:
```
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import LeaveOneOut
import collections
from tqdm import tqdm
from sklearn.model_selection import KFold
from sklearn.decomposition import PCA
```

Problem 1

In [ ]:
```
# open folder heart_data and read text files heart_trainSet.txt, heart_testSet.txt, and heart_trainLabels.txt, separate by comma n
with open(os.path.join('heart_data', 'heart_trainSet.txt')) as f:
    trainSet = pd.read_csv(f, sep=',', header=None)
with open(os.path.join('heart_data', 'heart_testSet.txt')) as f:
    testSet = pd.read_csv(f, sep=',', header=None)
with open(os.path.join('heart_data', 'heart_trainLabels.txt')) as f:
    trainLabels = pd.read_csv(f, sep=',', header=None)
```

In [ ]:
```
# shape of data
print('trainSet shape: ', trainSet.shape)
print('testSet shape: ', testSet.shape)
print('trainLabels shape: ', trainLabels.shape)
```

```
trainSet shape:  (243, 13)
testSet shape:  (27, 13)
trainLabels shape:  (243, 1)
```

In [ ]:
```
# build knn classifier
def knn_classifier(train, trainL, test, k):
    all_labels = []
    for i, x in test.iterrows():
        label = []
        for j, y in train.iterrows():
            distance = np.linalg.norm(x-y)
```

```
            label.append((distance, trainL.iloc[j,0]))
        label.sort()
        label = label[:k]
        label_count = collections.Counter([z[1] for z in label])
        max_label_count = label_count.most_common()[0][0]
        all_labels.append(max_label_count)
    return all_labels
```

In [ ]:
```
loo = LeaveOneOut()
loo.get_n_splits(trainSet)
loo_error = []
for k in tqdm(range(1,11)):
    error = 0
    for train_index, test_index in loo.split(trainSet):
        X_train, X_test = trainSet.iloc[train_index], trainSet.iloc[test_index]
        y_train, y_test = trainLabels.iloc[train_index], trainLabels.iloc[test_index]

        knn = knn_classifier(X_train.reset_index(drop = True), y_train.reset_index(drop = True), X_test.reset_index(drop = True),
        if knn[0] != y_test.iloc[0,0]:
            error += 1
    loo_error.append(error/len(trainSet))
```

```
100%|██████████| 10/10 [01:16<00:00,  7.69s/it]
```

In [ ]:
```
print(loo_error)
```

```
[0.22633744855967078, 0.22633744855967078, 0.1728395061728395, 0.18518518518518517, 0.18106995884773663, 0.18518518518518517, 0.1
6872427983539096, 0.18106995884773663, 0.18106995884773663, 0.18106995884773663]
```

In [ ]:
```
# print the minimum leave one out error and the corresponding k
print('The minimum leave one out error is', min(loo_error), 'and the corresponding k is', loo_error.index(min(loo_error))+1)
```

```
The minimum leave one out error is 0.16872427983539096 and the corresponding k is 7
```

In [ ]:
```
# use the best k to predict the labels of the test data points
knn = knn_classifier(trainSet, trainLabels, testSet, loo_error.index(min(loo_error))+1)
```

In [ ]:
```
# print the predicted labels of the test data points
print('The predicted labels of the test data points are', knn)
```

```
The predicted labels of the test data points are [-1, -1, 1, -1, 1, -1, -1, 1, -1, 1, 1, 1, -1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1,
1, -1, -1, 1]
```

Problem 2

In [ ]:

```python
with open(os.path.join('20newsgroups', 'train.data')) as f:
    train_data = np.loadtxt(f, dtype=int)
with open(os.path.join('20newsgroups', 'train.label')) as f:
    train_label = np.loadtxt(f, dtype=int)

with open(os.path.join('20newsgroups', 'test.data')) as f:
    test_data = np.loadtxt(f, dtype=int)
with open(os.path.join('20newsgroups', 'test.label')) as f:
    test_label = np.loadtxt(f, dtype=int)
```

In [ ]:

```python
# shape of data
print('train_data shape: ', train_data.shape)
print('train_label shape: ', train_label.shape)
print('test_data shape: ', test_data.shape)
print('test_label shape: ', test_label.shape)
```

```
train_data shape:  (1467345, 3)
train_label shape:  (11269,)
test_data shape:  (967874, 3)
test_label shape:  (7505,)
```

In [ ]:

```python
def naive_bayes_classifier(alpha, train, trainL, test, testL):
    # compute the class probabilities
    num_classes = len(np.unique(train_label))
    class_counts = np.bincount(train_label)[1:]  # Exclude the 0 count
    class_probs = class_counts / np.sum(class_counts)

    # compute the word probabilities with Laplace smoothing
    num_words = np.max(train[:, 1])
    word_counts = np.zeros((num_words, num_classes))
    for i in range(train.shape[0]):
        doc_id, word_id, count = train[i]
        class_id = trainL[doc_id - 1] - 1
        word_counts[word_id - 1, class_id] += count

    word_probs = (word_counts + alpha) / (np.sum(word_counts, axis=0) + alpha * num_words)

    # convert probabilities to log likelihoods
    log_class_probs = np.log(class_probs)
    log_word_probs = np.log(word_probs)

    # predict the class labels for the test documents
    predictions = []
    unique_test_docs = np.unique(test[:, 0])
    for doc_id in unique_test_docs:
        doc_data = test[test[:, 0] == doc_id]
        log_probs = np.array(log_class_probs)
```

```python
        for word_id, count in doc_data[:, 1:]:
            if word_id <= num_words:
                log_probs += count * log_word_probs[word_id - 1, :]
        pred_label = np.argmax(log_probs) + 1
        predictions.append(pred_label)

    # calculate the accuracy of the classifier
    accuracy = np.mean(np.array(predictions) == testL)
    return accuracy, predictions
```

In [ ]:

```python
# grid search for alpha
alphas = np.linspace(0.01, 2, 20)
best_alpha = alphas[0]
best_accuracy = 0

for alpha in tqdm(alphas):
    accuracy, _ = naive_bayes_classifier(alpha, train_data, train_label, test_data, test_label)
    print(f"Alpha: {alpha:.4f}, Accuracy: {accuracy:.4f}")
    if accuracy > best_accuracy:
        best_alpha = alpha
        best_accuracy = accuracy
print(f"Best alpha: {best_alpha:.4f}, Best accuracy: {best_accuracy:.4f}")
```

```
  5%|▌         | 1/20 [00:19<06:13, 19.63s/it]
Alpha: 0.0100, Accuracy: 0.8029
 10%|█         | 2/20 [00:39<05:53, 19.66s/it]
Alpha: 0.1147, Accuracy: 0.8057
 15%|█▌        | 3/20 [00:58<05:34, 19.67s/it]
Alpha: 0.2195, Accuracy: 0.8063
 20%|██        | 4/20 [01:19<05:17, 19.83s/it]
Alpha: 0.3242, Accuracy: 0.8036
 25%|██▌       | 5/20 [01:38<04:57, 19.83s/it]
Alpha: 0.4289, Accuracy: 0.8009
 30%|███       | 6/20 [01:58<04:37, 19.82s/it]
Alpha: 0.5337, Accuracy: 0.7985
 35%|███▌      | 7/20 [02:18<04:17, 19.81s/it]
Alpha: 0.6384, Accuracy: 0.7935
 40%|████      | 8/20 [02:39<04:03, 20.33s/it]
Alpha: 0.7432, Accuracy: 0.7912
 45%|████▌     | 9/20 [03:00<03:44, 20.42s/it]
Alpha: 0.8479, Accuracy: 0.7887
 50%|█████     | 10/20 [03:20<03:24, 20.42s/it]
Alpha: 0.9526, Accuracy: 0.7861
 55%|█████▌    | 11/20 [03:41<03:03, 20.36s/it]
```

```
Alpha: 1.0574, Accuracy: 0.7825
 60%|██████    | 12/20 [04:01<02:42, 20.37s/it]
Alpha: 1.1621, Accuracy: 0.7767
 65%|██████▌   | 13/20 [04:21<02:22, 20.38s/it]
Alpha: 1.2668, Accuracy: 0.7723
 70%|███████   | 14/20 [04:42<02:02, 20.34s/it]
Alpha: 1.3716, Accuracy: 0.7668
 75%|███████▌  | 15/20 [05:02<01:42, 20.44s/it]
Alpha: 1.4763, Accuracy: 0.7604
 80%|████████  | 16/20 [05:23<01:21, 20.49s/it]
Alpha: 1.5811, Accuracy: 0.7562
 85%|████████▌ | 17/20 [05:44<01:01, 20.52s/it]
Alpha: 1.6858, Accuracy: 0.7502
 90%|█████████ | 18/20 [06:04<00:40, 20.48s/it]
Alpha: 1.7905, Accuracy: 0.7464
 95%|█████████▌| 19/20 [06:25<00:20, 20.50s/it]
Alpha: 1.8953, Accuracy: 0.7422
100%|██████████| 20/20 [06:46<00:00, 20.30s/it]
Alpha: 2.0000, Accuracy: 0.7359
Best alpha: 0.2195, Best accuracy: 0.8063
```

In [ ]:

```
# use the best alpha to predict the labels of the test data points and save as a text file
best_accuracy, predictions = naive_bayes_classifier(best_alpha, train_data, train_label, test_data, test_label)
predictions = np.array(predictions)
np.savetxt('predicted_labels_with_best_alpha.txt', predictions, fmt='%d')
```

In [ ]:

```
print('The shape of the predicted labels is', predictions.shape, 'which matches the shape of the test data labels.')
```

```
The shape of the predicted labels is (7505,) which matches the shape of the test data labels
```

Problem 3

In [ ]:

```
with open(os.path.join('gisette', 'gisette_trainSet.txt')) as f:
    gisette_trainSet = np.loadtxt(f)
with open(os.path.join('gisette', 'gisette_testSet.txt')) as f:
    gisette_testSet = np.loadtxt(f)
with open(os.path.join('gisette', 'gisette_trainLabels.txt')) as f:
    gisette_trainLabels = np.loadtxt(f)
```

In [ ]:

```
# info of gisette datasets
print('The shape of gisette_trainSet is', gisette_trainSet.shape)
```

```
print('The shape of gisette_testSet is', gisette_testSet.shape)
print('The shape of gisette_trainLabels is', gisette_trainLabels.shape)
```

```
The shape of gisette_trainSet is (6000, 5000)
The shape of gisette_testSet is (1000, 5000)
The shape of gisette_trainLabels is (6000,)
```

On the original data

In [ ]:
```
# improved knn classifier from the previous question 1, here store the distances into matrix for faster computation
def most_common(lst):
    return max(set(lst), key=lst.count)
def euclidean(point, data):
    # Euclidean distance between points x & data
    return np.sqrt(np.sum((point - data)**2, axis=1))
class knn_classifer_new:
    def __init__(self, k, dist_metric=euclidean):
        self.k = k
        self.dist_metric = dist_metric
    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train
    def predict(self, X_test):
        neighbors = []
        for x in X_test:
            distances = self.dist_metric(x, self.X_train)
            y_sorted = [y for _, y in sorted(zip(distances, self.y_train))]
            neighbors.append(y_sorted[:self.k])
        return list(map(most_common, neighbors))
    def evaluate(self, X_test, y_test):
        y_pred = self.predict(X_test)
        accuracy = sum(y_pred == y_test) / len(y_test)
        return accuracy
```

I conducted 5-fold cross validation on the training set to find the best k value ranging from 1 to 15. The best k value is 3. Then I used the best k value to predict the test set. The accuracy is 0.8973.

In [ ]:
```
# 5-fold cross validation
kf = KFold(n_splits=5, shuffle=True, random_state=0)
ks = range(1, 16)
kf.get_n_splits(gisette_trainSet)
accuracy = np.zeros((len(ks), kf.get_n_splits()))
for i, k in enumerate(tqdm(ks)):
    for j, (train_index, test_index) in enumerate(tqdm(kf.split(trainSet), leave = True)):
        X_train, X_test = gisette_trainSet[train_index], gisette_trainSet[test_index]
        y_train, y_test = gisette_trainLabels[train_index], gisette_trainLabels[test_index]
```

```
        knn = knn_classifer_new(k=k)
        knn.fit(X_train, y_train)
        accuracy[i, j] = knn.evaluate(X_test, y_test)
```

```
5it [00:01,  2.84it/s][00:00<?, ?it/s]
5it [00:01,  2.82it/s][00:01<00:24,  1.76s/it]
5it [00:01,  2.76it/s][00:03<00:22,  1.77s/it]
5it [00:01,  3.31it/s][00:05<00:21,  1.79s/it]
5it [00:01,  4.14it/s][00:06<00:18,  1.68s/it]
5it [00:01,  4.28it/s][00:08<00:15,  1.51s/it]
5it [00:01,  4.09it/s][00:09<00:12,  1.39s/it]
5it [00:01,  4.11it/s][00:10<00:10,  1.34s/it]
5it [00:01,  4.24it/s][00:11<00:09,  1.30s/it]
5it [00:01,  4.18it/s][00:12<00:07,  1.26s/it]
5it [00:01,  3.85it/s] [00:14<00:06,  1.24s/it]
5it [00:01,  4.21it/s] [00:15<00:05,  1.26s/it]
5it [00:01,  4.32it/s] [00:16<00:03,  1.24s/it]
5it [00:01,  3.60it/s] [00:17<00:02,  1.22s/it]
5it [00:01,  2.81it/s] [00:19<00:01,  1.27s/it]
100%|██████████| 15/15 [00:20<00:00,  1.39s/it]
```

In [ ]:
```
accuracy
```

Out[ ]:
```
array([[0.93877551, 0.87755102, 0.83673469, 0.9375    , 0.875     ],
       [0.91836735, 0.87755102, 0.85714286, 0.9375    , 0.79166667],
       [0.93877551, 0.85714286, 0.85714286, 0.95833333, 0.875     ],
       [0.85714286, 0.87755102, 0.89795918, 0.89583333, 0.79166667],
       [0.87755102, 0.87755102, 0.85714286, 0.9375    , 0.83333333],
       [0.85714286, 0.87755102, 0.89795918, 0.9375    , 0.79166667],
       [0.87755102, 0.87755102, 0.85714286, 0.95833333, 0.83333333],
       [0.87755102, 0.87755102, 0.85714286, 0.9375    , 0.77083333],
       [0.89795918, 0.87755102, 0.85714286, 0.95833333, 0.8125    ],
       [0.85714286, 0.85714286, 0.83673469, 0.89583333, 0.77083333],
       [0.87755102, 0.85714286, 0.83673469, 0.91666667, 0.79166667],
       [0.87755102, 0.87755102, 0.83673469, 0.89583333, 0.77083333],
       [0.87755102, 0.87755102, 0.83673469, 0.89583333, 0.77083333],
       [0.85714286, 0.89795918, 0.85714286, 0.89583333, 0.77083333],
       [0.87755102, 0.87755102, 0.85714286, 0.91666667, 0.79166667]])
```

In [ ]:
```
# print the best k and the corresponding accuracy
print('The best k is', ks[np.argmax(np.mean(accuracy, axis=1))], 'and the corresponding accuracy is', np.max(np.mean(accuracy, axi
```

```
The best k is 3 and the corresponding accuracy is 0.8972789115646258
```

PCA

Here I used the PCA to reduce the dimension of the data. In order to choose the best `n_components`, I used 5-fold cross validation on the training set and reconstruction error as the metric.

To choose the best `n_components` range, I use cumulative variance and set the variance threshold to 0.9, meaning that I want to keep 90% of the variance.

In [ ]:
```
pca = PCA()
pca.fit(gisette_trainSet)
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
variance_threshold = 0.90
num_components = np.argmax(cumulative_variance > variance_threshold) + 1
print('The number of components that explain 90% of the variance is', num_components)
```

```
The number of components that explain 90% of the variance is 1321
```

I found the number of components that explan 90% of the variance is 1321. Then for the range of `n_components` in the cross validation, I used 1321 - 10 as the lower bound and 1321+11 as the upper bound.

In [ ]:
```
# conduct 5-fold to find the best n_components of PCA using reconstruction error as metric
kf = KFold(n_splits=5, shuffle=True, random_state=0)
n_components = range(num_components - 10, num_components + 11)
reconstruction_errors = np.zeros((len(n_components), kf.get_n_splits()))
for i, n in enumerate(tqdm(n_components)):
    for j, (train_index, test_index) in enumerate(tqdm(kf.split(gisette_trainSet), leave = True)):
        X_train, X_test = gisette_trainSet[train_index], gisette_trainSet[test_index]
        pca = PCA(n_components=n)
        pca.fit(X_train)
        X_test_transformed = pca.transform(X_test)
        X_test_reconstructed = pca.inverse_transform(X_test_transformed)
        reconstruction_error = np.mean((X_test - X_test_reconstructed)**2)
        reconstruction_errors[i, j] = reconstruction_error
print('mean reconstruction errors are', np.mean(reconstruction_errors, axis=1))
print('best n_components is', n_components[np.argmin(np.mean(reconstruction_errors, axis=1))])
```

```
5it [00:59, 11.96s/it][00:00<?, ?it/s]
5it [00:55, 11.20s/it][00:59<19:56, 59.81s/it]
5it [00:57, 11.52s/it][01:55<18:13, 57.57s/it]
5it [00:59, 11.91s/it][02:53<17:16, 57.58s/it]
5it [00:57, 11.59s/it][03:52<16:32, 58.36s/it]
5it [00:57, 11.51s/it][04:50<15:31, 58.22s/it]
5it [00:57, 11.49s/it][05:48<14:30, 58.00s/it]
5it [00:57, 11.40s/it][06:45<13:29, 57.82s/it]
5it [00:59, 11.92s/it][07:42<12:28, 57.56s/it]
5it [00:58, 11.72s/it][08:42<11:38, 58.20s/it]
5it [00:58, 11.68s/it] [09:41<10:41, 58.33s/it]
5it [00:57, 11.51s/it] [10:39<09:43, 58.36s/it]
5it [00:58, 11.69s/it] [11:37<08:42, 58.11s/it]
5it [00:58, 11.64s/it] [12:35<07:45, 58.22s/it]
5it [01:04, 12.80s/it] [13:33<06:47, 58.21s/it]
5it [01:00, 12.03s/it] [14:37<05:59, 59.96s/it]
5it [00:58, 11.69s/it] [15:38<05:00, 60.03s/it]
5it [00:57, 11.44s/it] [16:36<03:58, 59.56s/it]
5it [01:02, 12.50s/it] [17:33<02:56, 58.85s/it]
5it [01:05, 13.00s/it] [18:36<01:59, 59.94s/it]
5it [00:59, 11.95s/it] [19:41<01:01, 61.47s/it]
100%|██████████| 21/21 [20:40<00:00, 59.09s/it]
mean reconstruction errors are [0.05371051 0.05365974 0.05360173 0.05354236 0.05348873 0.05344363
 0.05337926 0.05333204 0.05328568 0.05321311 0.05316785 0.05313065
 0.05306906 0.05301959 0.05296975 0.05291237 0.05284577 0.05280097
 0.05275313 0.05268477 0.05263386]
best n_components is 1331
```

I found the best `n_components` is 1331. Then I used the best `n_components` to transform the data.

In [ ]:

```python
# use the best n_components to transform the trainSet, testSet
pca = PCA(n_components=1331)
pca.fit(gisette_trainSet)
gisette_trainSet_transformed = pca.transform(gisette_trainSet)
gisette_testSet_transformed = pca.transform(gisette_testSet)
```

Then I used the reduced dimention data for KNN classifier and use 5-fold cross validation to find the best `k` value from the same range.

In [ ]:

```python
# k-fold cross validation to find the best k
kf = KFold(n_splits=5, shuffle=True, random_state=0)
ks = range(1, 16)
kf.get_n_splits(gisette_trainSet_transformed)
accuracy = np.zeros((len(ks), kf.get_n_splits()))
for i, k in enumerate(tqdm(ks)):
    for j, (train_index, test_index) in enumerate(tqdm(kf.split(gisette_trainSet_transformed), leave = True)):
        X_train, X_test = gisette_trainSet_transformed[train_index], gisette_trainSet_transformed[test_index]
```

```
        y_train, y_test = gisette_trainLabels[train_index], gisette_trainLabels[test_index]
        knn = knn_classifer_new(k=k)
        knn.fit(X_train, y_train)
        accuracy[i, j] = knn.evaluate(X_test, y_test)
```

```
5it [03:30, 42.13s/it][00:00<?, ?it/s]
5it [03:22, 40.59s/it][03:30<49:08, 210.64s/it]
5it [03:23, 40.75s/it][06:53<44:39, 206.11s/it]
5it [03:21, 40.30s/it][10:17<41:00, 205.04s/it]
5it [03:23, 40.79s/it][13:38<37:20, 203.64s/it]
5it [03:28, 41.62s/it][17:02<33:57, 203.76s/it]
5it [03:29, 41.93s/it][20:30<30:47, 205.24s/it]
5it [03:22, 40.47s/it][24:00<27:33, 206.68s/it]
5it [03:26, 41.26s/it][27:22<23:57, 205.31s/it]
5it [03:27, 41.58s/it][30:49<20:33, 205.62s/it]
5it [03:23, 40.76s/it] [34:17<17:11, 206.33s/it]
5it [03:30, 42.12s/it] [37:41<13:42, 205.56s/it]
5it [03:25, 41.11s/it] [41:11<10:21, 207.10s/it]
5it [03:24, 40.92s/it] [44:37<06:53, 206.63s/it]
5it [03:27, 41.40s/it] [48:01<03:26, 206.02s/it]
100%|██████████| 15/15 [51:28<00:00, 205.92s/it]
```

In [ ]:
```
accuracy
```

Out[ ]:
```
array([[0.9575    , 0.955     , 0.96666667, 0.95916667, 0.95666667],
       [0.95083333, 0.94166667, 0.9525    , 0.94833333, 0.94333333],
       [0.9725    , 0.9625    , 0.96416667, 0.9675    , 0.96583333],
       [0.955     , 0.95166667, 0.955     , 0.96166667, 0.95416667],
       [0.9675    , 0.96      , 0.96416667, 0.96666667, 0.96583333],
       [0.9575    , 0.95666667, 0.9525    , 0.965     , 0.9575    ],
       [0.97083333, 0.965     , 0.96      , 0.965     , 0.96833333],
       [0.96      , 0.96583333, 0.955     , 0.95916667, 0.96      ],
       [0.96833333, 0.9625    , 0.95916667, 0.95833333, 0.96666667],
       [0.96416667, 0.955     , 0.9575    , 0.95833333, 0.95916667],
       [0.965     , 0.95916667, 0.96083333, 0.95833333, 0.96583333],
       [0.95833333, 0.955     , 0.95583333, 0.95583333, 0.96      ],
       [0.96416667, 0.96083333, 0.955     , 0.95916667, 0.96416667],
       [0.96083333, 0.95583333, 0.9575    , 0.9575    , 0.96      ],
       [0.96166667, 0.9575    , 0.96      , 0.96083333, 0.965     ]])
```

In [ ]:
```
# print the best k and the corresponding accuracy
print('The best k is', ks[np.argmax(np.mean(accuracy, axis=1))], 'and the corresponding accuracy is', np.max(np.mean(accuracy, axi
```

```
The best k is 3 and the corresponding accuracy is 0.9665000000000001
```

The best `k` value is 3, and the accuracy is 0.97, which is higher than the accuracy of the original data.

Problem 4

proof The median distance from the origin to its nearest neighbor is denoted by α, such that $P(D \geqslant \alpha) = \frac{1}{2}$, where D represents the euclidean distance from the origin to its closest point in the unit ball with p dimensions. Let $F(\alpha)$ denote the CDF of D, and let N be the number of data points in the unit ball.

By definition, we have $P(D \geqslant \alpha) = 1 - F(\alpha)^N$, where $1 - F(\alpha)^N$ represents the probability that all N data points are farther than α from the origin. Therefore, we can express the CDF of the smallest distance from the origin to its nearest neighbor as:

$P(D \leqslant \alpha) = 1 - P(D > \alpha) = 1 - (1 - F(\alpha)^N) = 1 - (1 - \alpha^p)^N$

The median distance is the value of α such that $P(D \leqslant \alpha) \geqslant \frac{1}{2}$ and $P(D > \alpha) \geqslant \frac{1}{2}$. Thus, we have:

$\frac{1}{2} = P(D > \alpha) = 1 - F(\alpha)^N$

Solving for α, we get:

$\alpha^p = 1 - \frac{1}{2^N}$

Taking the p th root of both sides, we obtain:

$\alpha = 1 - (2^{-\frac{1}{N}})^{\frac{1}{p}}$

$\square$