2IMW30 - FOUNDATION OF DATA MINING
Assignment 1c Report

Duy Pham (0980384, d.pham.duy@student.tue.nl)
Mazen Aly(0978251, m.aly@student.tue.nl)

(Both students contributed equally in this assignment)

EXERCISE 1

In this assignment, we implement the Random Projection Algorithm for reducing the number of dimensions in the MNIST dataset. The whole source code of the project can be found in the Github repository `https://github.com/MazenAly/datamining`.

The idea of random projections is simply if we have points in a vector space that are of a sufficiently high dimension, then these points can be projected into a suitably lower dimensional space in a way which approximately preserves the distances between the points. We achieve this by randomly generate a $k \times d$ matrix with the assignment stated coefficients.

```
def create_matrix(d, k)
    mtx_val = 1/sqrt(d)
    prob = 0
    mtx = np.zeros((d, k))
    for i in range(0,d):
        for j in range(0,k):
            random_number = np.random.random_sample()
            if random_number <= 0.5:
                mtx[i,j] = mtx_val
            else:
                mtx[i,j] = -mtx_val
    return mtx
```

Then we evaluate how well the Euclidean distance is preserved for the first 20 instances of the dataset by plotting the distortion for all the $\binom{20}{2}$ pairs.

We expect the distortion values to be between 0 and 1 (as proved in exercise 2). Besides, when k increases, the number dimensions approaches the original dimension d. Thus, the distortion value should approach 1.

Our expectations are confirmed by our experiments as in the below figures.

In figure 1, we can see that the most frequent value of distortion values is around 0.25 when $k = 50$.

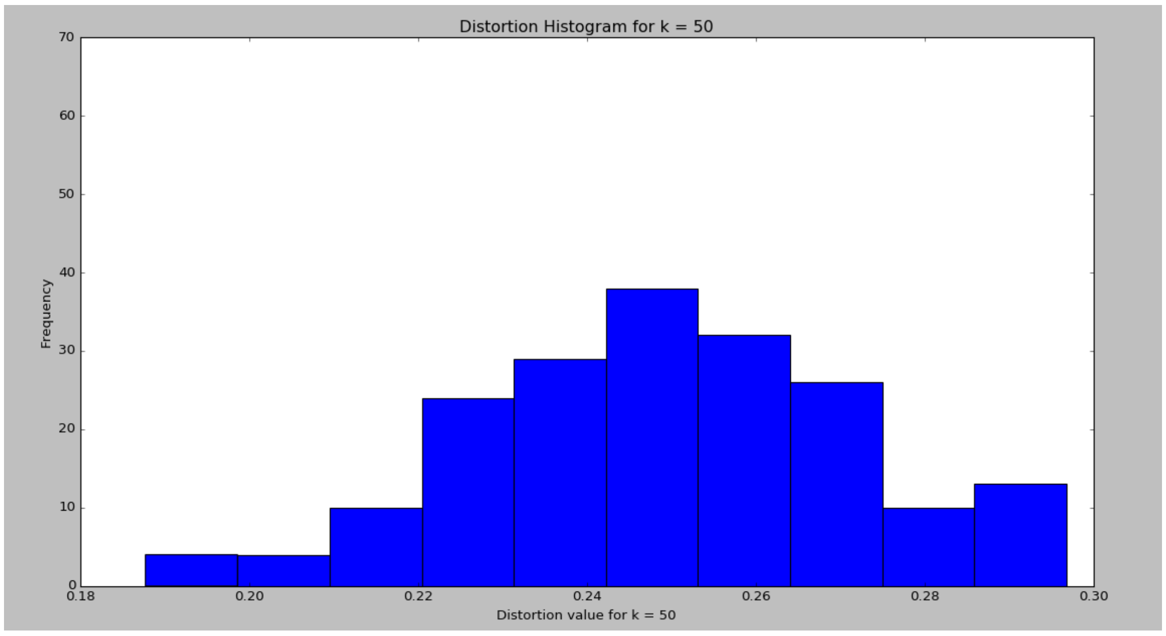


Figure 1: Histogram of the distortion values with k = 50

In figure 2, the most frequent value of distortion values when $k = 100$ is around 0.35.

As we predict, we can see that the distortion value increases to 0.79 when $k = 500$ as in figure 3.

In addition, we change our 1-NN implementation of Assignment 1a so that it uses Euclidean Distance instead of Cosine Similarity, as in the below function

```
def euclidean_distance(v1, v2):
    diff = np.subtract(v1, v2)
    return np.linalg.norm(diff)
```

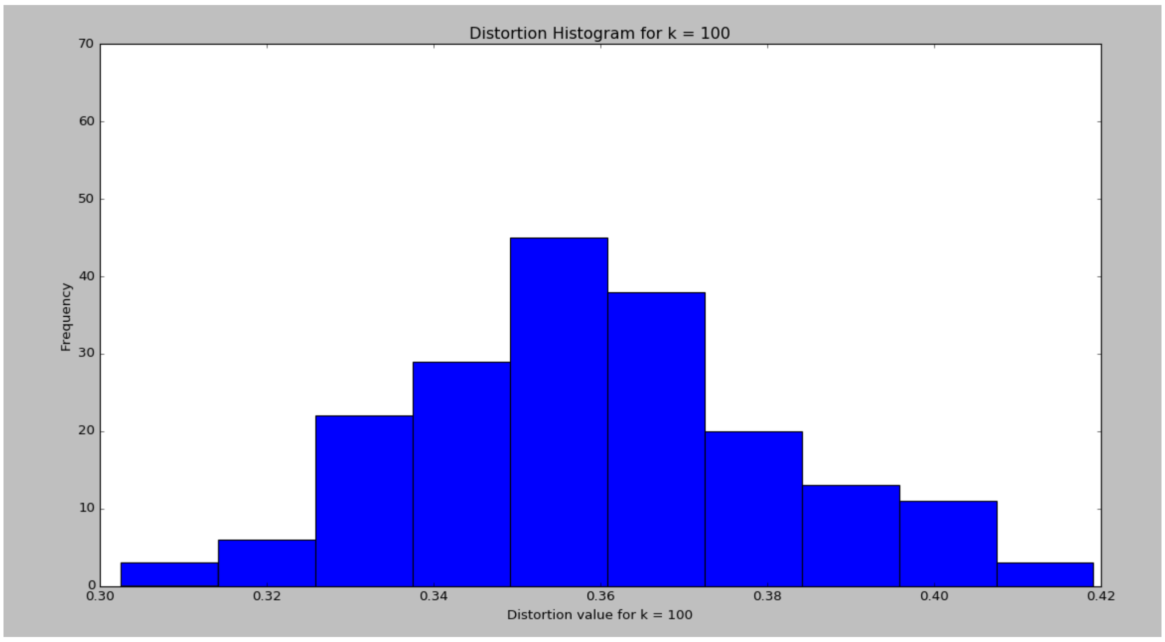


Figure 2: Histogram of the distortion values with k = 100

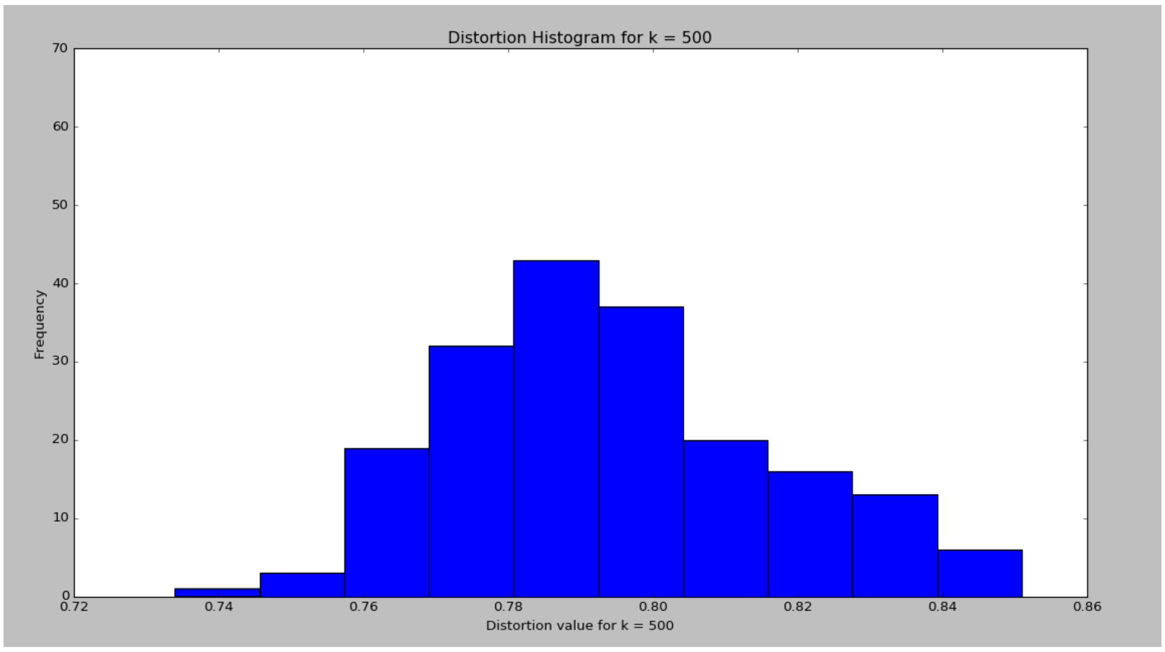


Figure 3: Histogram of the distortion values with k = 500

We run our implementation with and without random projection. We scale our vectors after the projection step to partially compensate for the distortion using a scaling factor $\frac{\sqrt{d}}{\sqrt{k}}$. Note that we run the implementation for 1000 test cases, as it takes a lot of time to experiment the whole 10,000 test cases.

We perform 4 experiments, with and without Random Projections. When using Random Projections, we experiment 3 times with $k = 50, 100, 500$ to compare the results. The confusion matrices are shown in table 1, 2, 3, and 4.

1-NN without Random Projections

```
Accuracy of the NN classifier is :  95.57 %
Average Runtime: 0.27 seconds for each instance

Class  0 :
```

Table 1: The final confusion matrix without RF

Predicted-TrueLabel	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
0	84	0	1	0	0	0	1	0	1	0	87
1	0	125	0	0	1	0	0	2	0	0	128
2	0	1	110	1	0	0	0	0	0	0	112
3	0	0	0	100	0	2	0	2	2	1	107
4	0	0	1	0	104	1	1	1	0	0	108
5	1	0	0	3	0	84	0	0	3	0	91
6	0	0	0	0	0	0	85	0	0	0	85
7	0	0	4	2	0	0	0	93	0	2	101
8	0	0	0	1	0	0	0	0	82	1	84
9	0	0	0	0	5	0	0	1	1	90	97
Total	85	126	116	107	110	87	87	99	89	94	

```
Precision: 0.965517241379
Recall: 0.988235294118

Class 1 :
Precision: 0.9765625
Recall: 0.992063492063

Class 2 :
Precision: 0.982142857143
Recall: 0.948275862069

Class 3 :
Precision: 0.934579439252
Recall: 0.934579439252

Class 4 :
Precision: 0.962962962963
Recall: 0.945454545455

Class 5 :
Precision: 0.923076923077
Recall: 0.965517241379

Class 6 :
Precision: 1.0
Recall: 0.977011494253

Class 7 :
Precision: 0.920792079208
Recall: 0.939393939394

Class 8 :
Precision: 0.97619047619
Recall: 0.921348314607

Class 9 :
Precision: 0.927835051546
Recall: 0.957446808511
```

1-NN with Random Projections, $k = 50$

```
Accuracy of the NN classifier is :  92.3 %
Average Runtime: 0.23 seconds for each instance

Class  0 :
Precision: 0.955056179775
Recall: 1.0

Class  1 :
Precision: 0.932835820896
Recall: 0.992063492063

Class  2 :
Precision: 0.951923076923
Recall: 0.853448275862

Class  3 :
Precision: 0.92380952381
Recall: 0.906542056075

Class  4 :
Precision: 0.960784313725
Recall: 0.890909090909

Class  5 :
Precision: 0.873684210526
Recall: 0.954022988506

Class  6 :
Precision: 0.965517241379
Recall: 0.965517241379

Class  7 :
Precision: 0.92
Recall: 0.929292929293

Class  8 :
Precision: 0.890243902439
Recall: 0.820224719101

Class  9 :
Precision: 0.852941176471
Recall: 0.925531914894
```

For 1-NN with Random Projections of $k = 100$

```
Accuracy of the NN classifier is :  94.0 %
Average Runtime: 0.25 seconds for each instance

Class  0 :
Precision: 0.976744186047
Recall: 0.988235294118

Class  1 :
Precision: 0.9765625
Recall: 0.992063492063

Class  2 :
Precision: 0.972972972973
```

Table 2: The final confusion matrix with $k = 50$

Predicted-TrueLabel	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
0	85	0	1	0	0	0	2	0	1	0	89
1	0	125	2	1	1	0	0	3	1	1	134
2	0	1	99	0	0	0	0	0	4	0	104
3	0	0	2	97	0	1	0	2	1	2	105
4	0	0	0	0	98	0	0	0	1	3	102
5	0	0	1	7	0	83	0	0	4	0	95
6	0	0	2	0	0	0	84	0	1	0	87
7	0	0	2	1	0	2	1	92	1	1	100
8	0	0	7	1	1	0	0	0	73	0	82
9	0	0	0	0	10	1	0	2	2	87	102
Total	85	126	116	107	110	87	87	99	89	94	

```
Recall: 0.931034482759

Class 3 :
Precision: 0.924528301887
Recall: 0.915887850467

Class 4 :
Precision: 0.960784313725
Recall: 0.890909090909

Class 5 :
Precision: 0.870967741935
Recall: 0.931034482759

Class 6 :
Precision: 0.965909090909
Recall: 0.977011494253

Class 7 :
Precision: 0.932038834951
Recall: 0.969696969697

Class 8 :
Precision: 0.95
Recall: 0.85393258427

Class 9 :
Precision: 0.864077669903
Recall: 0.946808510638
```

For 1-NN with Random Projections of $k = 500$

```
Accuracy of the NN classifier is :  95.6 %
Average Runtime: 0.27 seconds for each instance

Class 0 :
Precision: 0.943820224719
Recall: 0.988235294118

Class 1 :
Precision: 0.984251968504
Recall: 0.992063492063
```

Table 3: The final confusion matrix with RF, $k = 100$

Predicted-TrueLabel	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
0	84	0	1	0	0	0	1	0	0	0	86
1	0	125	1	0	0	0	0	1	0	1	128
2	0	1	108	1	0	0	0	0	1	0	111
3	0	0	0	98	0	3	0	1	4	0	106
4	0	0	1	0	98	0	1	0	0	2	102
5	0	0	0	3	0	81	0	0	8	1	93
6	1	0	0	0	2	0	85	0	0	0	88
7	0	0	4	2	1	0	0	96	0	0	103
8	0	0	1	1	0	1	0	0	76	1	80
9	0	0	0	2	9	2	0	1	0	89	103
Total	85	126	116	107	110	87	87	99	89	94	

```
Class  2 :
Precision: 0.981981981982
Recall: 0.939655172414

Class  3 :
Precision: 0.942857142857
Recall: 0.92523364486

Class  4 :
Precision: 0.963963963964
Recall: 0.972727272727

Class  5 :
Precision: 0.893617021277
Recall: 0.965517241379

Class  6 :
Precision: 0.977011494253
Recall: 0.977011494253

Class  7 :
Precision: 0.949494949495
Recall: 0.949494949495

Class  8 :
Precision: 0.963414634146
Recall: 0.887640449438

Class  9 :
Precision: 0.947368421053
Recall: 0.957446808511
```

We can see that the performance of the classifier with random projections of $k = 50$ and $k = 100$ is lower than the performance without projections as this huge dimensionality reduction causes a big loss of information. But we can see the performance is improving with the increment of k. For $k = 500$, the performance become slightly better than the classifier without projections. The cause could be some noise and outliers were reduced by our dimensionality reduction, and the value of k is close to the original d which preserves most of the information.

Regarding the runtime, the dimensionality reduction does not show a huge difference among the different settings. It might be because the problem is still not complex enough. If we have thousands of dimensions, then the difference can be clearer. The fastest one is around 0.2 seconds for an instance, and the slowest one (without Random Projections) is around 0.3 seconds for an instance.

Table 4: The final confusion matrix with RF, $k = 500$

Predicted-TrueLabel	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
0	84	0	2	0	0	1	1	0	1	0	89
1	0	125	0	0	0	0	0	2	0	0	127
2	0	1	109	0	0	0	0	0	1	0	111
3	0	0	0	99	0	1	0	2	2	1	105
4	0	0	0	0	107	1	1	1	0	1	111
5	1	0	0	4	0	84	0	0	5	0	94
6	0	0	2	0	0	0	85	0	0	0	87
7	0	0	3	2	0	0	0	94	0	0	99
8	0	0	0	1	0	0	0	0	79	2	82
9	0	0	0	1	3	0	0	0	1	90	95
Total	85	126	116	107	110	87	87	99	89	94	

EXERCISE 2

Since the nearest neighbor in F of a point P in R^d is the projection of P onto F, a linear mapping that maps a point to its nearest neighbor in a subspace can be simulated by a rotation followed by an orthogonal projection.

Let P_1 and P_2 be 2 points in R^d, M be the rotation matrix, and P be the projection matrix.

Let $Q_1 = M \cdot P_1$, $Q_2 = M \cdot P_2$, $P_1' = P \cdot Q_1$, and $P_2' = P \cdot Q_2$. So P_1' and P_2' are the resulting points.

We know that Rotation preserves the Euclidean distance, so $Q_1Q_2 = P_1P_2$.

Because P contains only 0 and 1, multiplying P to Q_1 and Q_2 preserves several features of Q_1 and Q_2, and set everything else to 0. Thus, $(P_1' - P_2') \leq (Q_1 - Q_2)$, which gives $(P_1' - P_2') \leq (P_1 - P_2)$.

This completes the proof.

EXERCISE 3

a

Let $d(x, y)$ be the distance from a point $P(x, y)$ to the center of the circle. Then the unit circle equation is $d(x, y) = 1$.

- When $p = 1$, we have $x + y = 1$
- When $p = 2$, we have $\sqrt{x^2 + y^2} = 1$
- When $p = 10$, we have $(x^{10} + y^{10})^{\frac{1}{10}} = 1$
- When $p = \infty$, we have $max(x, y) = 1$

Plotting the graphs of those equations, we have the unit circles, shown in figure 4.

b

We can see that the 2 unit circles (the 2 squares indeed) are quite similar. The differences are the angle ($pi/4$) and the length of the edge (2 and $\sqrt{2}$). Thus, we can preserve the distance by doing a rotation by an angle of $pi/4$ followed by a scaling by a factor of$\sqrt{2}$.

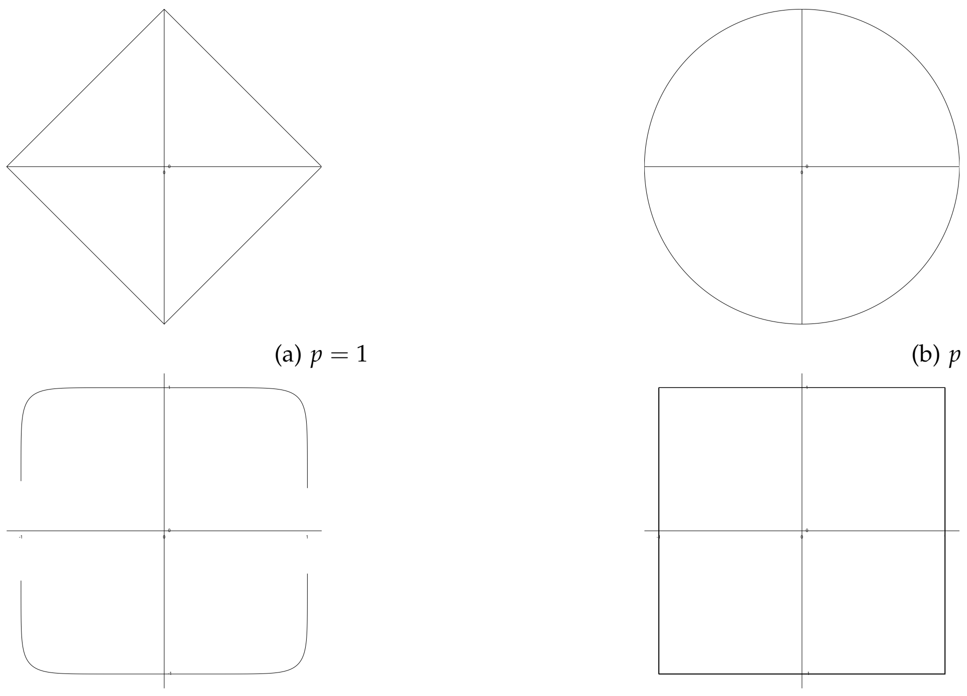
The rotation matrix is:

$$\begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}$$

So the mapping function can be done by multiplying the following matrix to the original data:

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

Now we prove that the mapping function actually preserves the distance. As we know, the difference between 2 vectors is also a vector. Let $p = (a, b)$ be a difference between 2 vectors in space. After the transformation, $p' = (a - b, a + b)$. We have:



(a) $p = 1$ (b) $p = 2$

(c) $p = 10$ (d) $p = infi$

Figure 4: Unit Circles

$$\begin{aligned}||p||_1 &= |a| + |b| \\ ||p'||_\infty &= max(|a-b|, |a+b|) = |a| + |b|\end{aligned}$$

So $||p||_1 = ||p'||_\infty$. This completes the proof.