Cem Daloğlu
21602161 / 12.04.2021

EEE 361 Homework-3 Report

In this homework report, application of the Johnson-Lindenstrauss Lemma will be discussed. $J, m, n, k, \epsilon, v_i, v_{av}$ will be used as the same abbreviations as in the homework assignment report. Random projections J are applied as stated in the assignment and for different m, ϵ and n values the following k values are obtained;

```
m = 10^3, n = 10^3, epsilon = 0.1, k = 5527
m = 10^3, n = 10^3, epsilon = 0.3, k = 615
m = 10^3, n = 10^3, epsilon = 0.7, k = 113
m = 10^3, n = 10^3, epsilon = 0.9, k = 69

m = 10^3, n = 10^2, epsilon = 0.1, k = 3685
m = 10^3, n = 10^2, epsilon = 0.3, k = 410
m = 10^3, n = 10^2, epsilon = 0.7, k = 76
m = 10^3, n = 10^2, epsilon = 0.9, k = 46

m = 10^3, n = 10, epsilon = 0.1, k = 1843
m = 10^3, n = 10, epsilon = 0.3, k = 205
m = 10^3, n = 10, epsilon = 0.7, k = 38
m = 10^3, n = 10, epsilon = 0.9, k = 23

time: 128.31620621681213
```

```
m = 5*10^3, n = 5*10^3, epsilon = 0.1, k = 6814
m = 5*10^3, n = 5*10^3, epsilon = 0.3, k = 758
m = 5*10^3, n = 5*10^3, epsilon = 0.7, k = 140
m = 5*10^3, n = 5*10^3, epsilon = 0.9, k = 85

m = 5*10^3, n = 5*10^2, epsilon = 0.1, k = 4972
m = 5*10^3, n = 5*10^2, epsilon = 0.3, k = 553
m = 5*10^3, n = 5*10^2, epsilon = 0.7, k = 102
m = 5*10^3, n = 5*10^2, epsilon = 0.9, k = 62

m = 5*10^3, n = 50, epsilon = 0.1, k = 3130
m = 5*10^3, n = 50, epsilon = 0.3, k = 348
m = 5*10^3, n = 50, epsilon = 0.7, k = 64
m = 5*10^3, n = 50, epsilon = 0.9, k = 39

time: 2952.003109931946
```

```
m = 10^4, n = 10^4, epsilon = 0.1, k = 7369
m = 10^4, n = 10^4, epsilon = 0.3, k = 819
m = 10^4, n = 10^4, epsilon = 0.7, k = 151
m = 10^4, n = 10^4, epsilon = 0.9, k = 91

m = 10^4, n = 10^3, epsilon = 0.1, k = 5527
m = 10^4, n = 10^3, epsilon = 0.3, k = 615
m = 10^4, n = 10^3, epsilon = 0.7, k = 113
m = 10^4, n = 10^3, epsilon = 0.9, k = 69

m = 10^4, n = 10^2, epsilon = 0.1, k = 3685
m = 10^4, n = 10^2, epsilon = 0.3, k = 410
m = 10^4, n = 10^2, epsilon = 0.7, k = 76
m = 10^4, n = 10^2, epsilon = 0.9, k = 46

time: 34252.70426130295
```

Figure 1: k values for $m = \{10^3, 5 * 10^3, 10^4\}$ and $\epsilon = \{0.1, 0.3, 0.7, 0.9\}$ for $n = \{m, \frac{m}{10}, \frac{m}{100}\}$

Note that, since the given m values are too large, matrices with size $\frac{m}{10}$ is used. k is inversely proportional with ϵ from the given inequality in the assignment and it can be observed in Figure 1 as well. When a small k is chosen, the projection matrix becomes larger than the original matrix, for instance, when $m = n = 10^3$, the k is larger than 10^3. It would be better if each computations time could be seen but unfortunately the overall time for each m is only printed. As it can be seen in Figure 1, for 10^4 it took approximately 10 hours to check the all distances (condition 1 in the assignment) that is the reason behind changing input matrix sizes. Afterwards, the given tasks are implemented and the following plots are obtained for the first task.

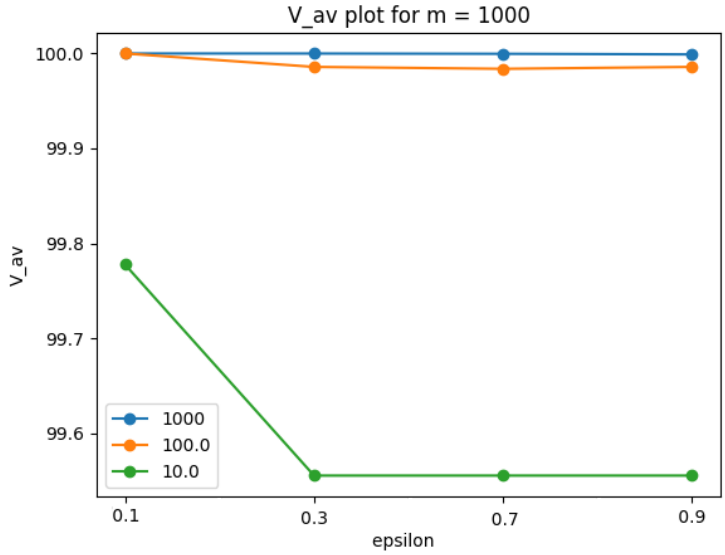

Figure 2: v_{av} plot for $m = 10^3$ and $\epsilon = \{0.1, 0.3, 0.7, 0.9\}$ for different n values

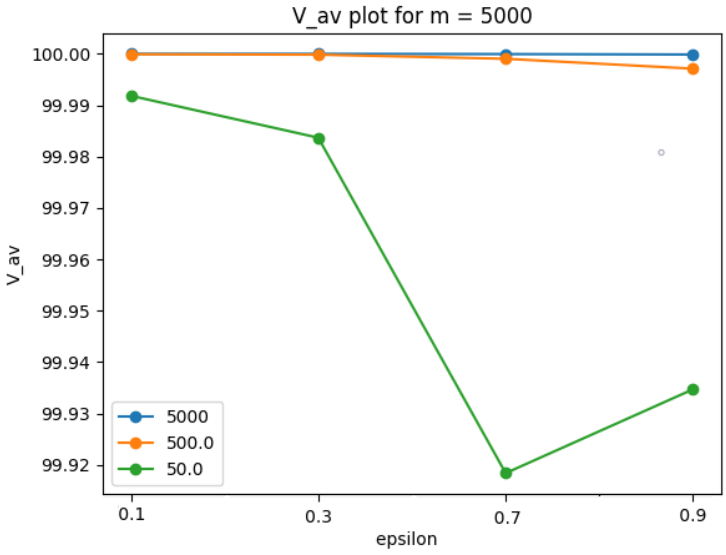

Figure 3: v_{av} plot for $m = 5 * 10^3$ and $\epsilon = \{0.1, 0.3, 0.7, 0.9\}$ for different n values

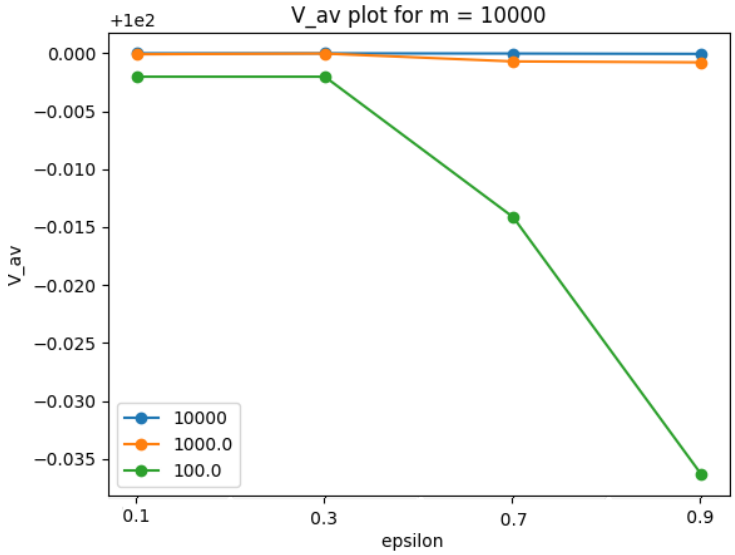

Figure 4: v_{av} plot for $m = 10^4$ and $\epsilon = \{0.1, 0.3, 0.7, 0.9\}$ for different n values

As it can be seen in Figures 2, 3, and 4, as n increases the difference (v_{av}) between the original vectors (x) and the random projection vectors (J) decreases. This is the reason that the lowest percentage of cases that condition (1) holds is obtained for $m = 10^3 \; and \; n = 10$. For the same reason the highest percentage of cases that condition (1) holds is obtained for $m = 10^4 \; and \; n = 10^4$. From the Johnson-Lindenstrauss Lemma, as ϵ decreases projection dimension k increases. However, from the Figures 2, 3, and 4, there is no significant impact of k is observed in the percentages of cases that condition (1) holds for large sizes. By looking at the green lines of the above plots, it can be said that projection matrices with smaller ϵ or larger k has greater percentage than larger ϵ or smaller k. In other words, projection matrices dimension (k) and input matrix size (m, n) is proportional, ϵ is inversely proportional with percentage value (v_{av}).

The following plots are obtained for the second task. Note that, in the figures below, all legends have typo, rather than numbers they can be thought as $m = n$ for blue, $n = m/10$ for yellow, and n$n = m/100$ for the green lines.

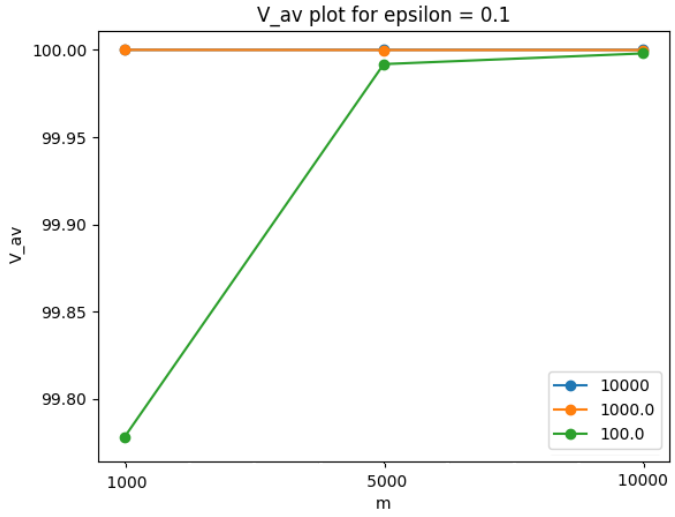

Figure 5: v_{av} plot for $\epsilon = 0.1$ and $m = \{10^3, 5 * 10^3, 10^4\}$ for different n values

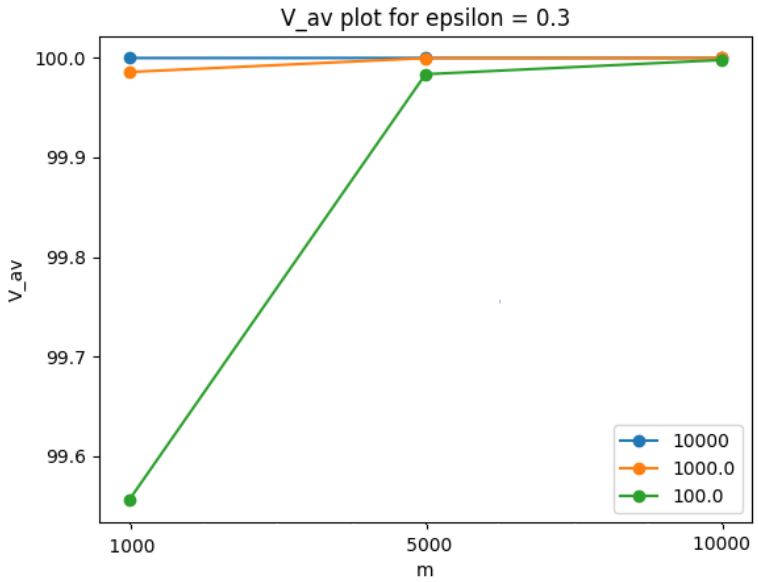

Figure 6: v_{av} plot for $\epsilon = 0.3$ and $m = \{10^3, 5 * 10^3, 10^4\}$ for different n values

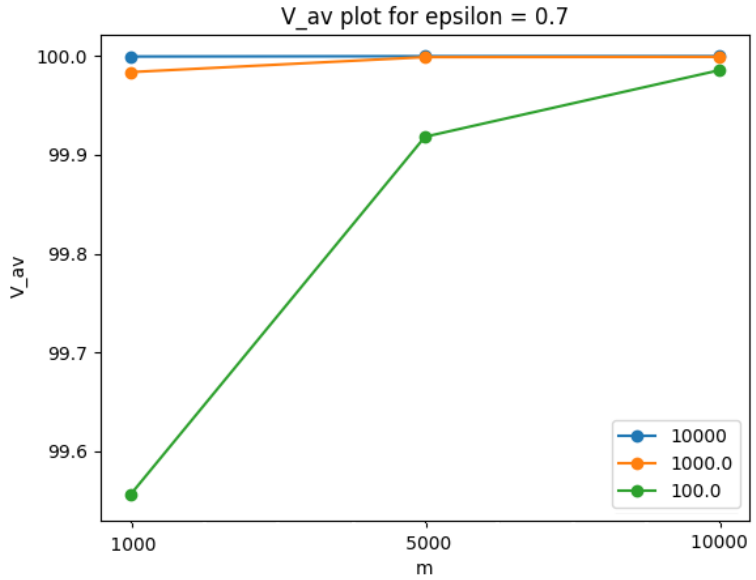

Figure 7: v_{av} plot for $\epsilon = 0.7$ and $m = \{10^3, 5 * 10^3, 10^4\}$ for different n values

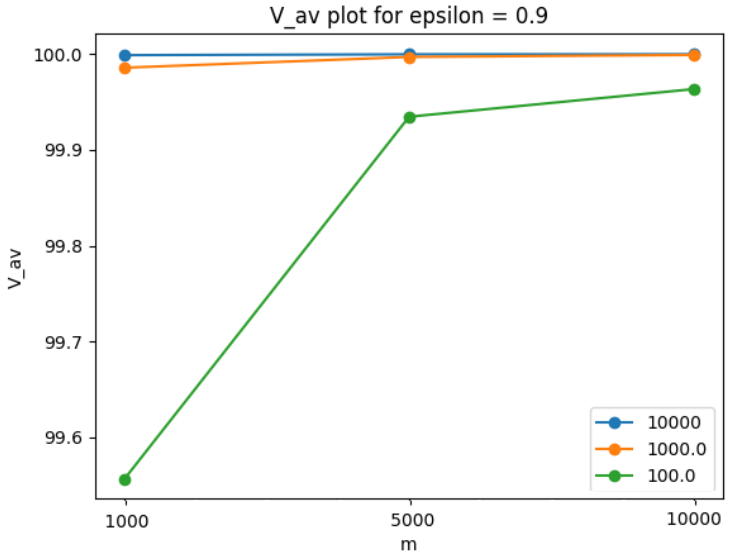

Figure 8: v_{av} plot for $\epsilon = 0.9$ and $m = \{10^3, 5 * 10^3, 10^4\}$ for different n values

As it can be seen in Figures 5, 6, 7, and 8, as $m\ and\ n$ increases the difference (v_{av}) between the original vectors (x) and the random projection vectors (J) decreases. By looking at the green lines of the above plots, it can be said that matrices with smaller m, n has smaller percentage rate than matrices with larger m, n. Also, it can be said that since the highest percentage rate is obtained in $\epsilon = 0.1$ in Figure 5, ϵ is inversely proportional with the percentage rate. To sum up, the same conclusion made for the previous figures can be made for the above figures as well. The projection matrices dimension (k) and input matrix size (m, n) is proportional, ϵ is inversely proportional with percentage value (v_{av}). The general trend of the percentage rate is going to 100% as the size of the input matrix increases and ϵ decreases. However, for smaller ϵ, projection matrix size also increases and sometimes it is larger than the input matrix size. The purpose of the J-L lemma is to reduce the input matrix dimension with a small amount. In order to achieve that purpose, ϵ can be chosen according to what error rate is tolerable for the system that will use the projection matrix. If error tolerance is high then, a high ϵ can be chosen, it would reduce the time to process the matrix.

Appendix

```
import matplotlib.pyplot as plt
import cupy as cp
import numpy as np
import math
import time

rand_numbers = np.random.default_rng(seed=0)


def x_generator(m, n):
    return rand_numbers.random((m, n))


def k_generator(X, eps):
```

```
    m, n = X.shape
    return 8 * np.log(n) / (eps ** 2)


def J_generator(X, k):
    m, _ = X.shape
    return rand_numbers.normal(0, 1 / math.sqrt(k), size=(m, k))


def condition_check(X, J, eps):
    _, n = X.shape
    check = n * n - n
    gpu_x = cp.asarray(X)
    x_gpu_norm = gpu_x.T.dot(gpu_x)
    x_norm = cp.asnumpy(x_gpu_norm)
    V = J.T.dot(X)
    Jx = V.T.dot(V)
    for i in range(n):
        for j in range(n):
            if i != j:
                dist = Jx[i, i] - 2 * Jx[i, j] + Jx[j, j]
                norm_ = x_norm[i, i] - 2 * x_norm[i, j] + x_norm[j, j]
                if not (dist <= (1 + eps) * norm_) & (dist >= (1 - eps) *
norm_):
                    check = check - 1
    check = (check / (n * n - n)) * 100
    return check


m = [1000, 5000, 10000]
eps = [0.1, 0.3, 0.7, 0.9]
n_arr = [1, 1/10, 1/100]
vi = np.zeros(10)
vav = np.zeros(36)

for i in range(len(m)):
    start_time = time.time()
    for n_i in range(len(n_arr)):
        n = int(m[i] * n_arr[n_i])
        X = x_generator(m[i], n)
        for e in range(len(eps)):
            k = k_generator(X, eps[e])
            k = int(k) + 1
            print(f'k = {k}')
            for realization in range(10):
                J = J_generator(X, k)
                vi[realization] = condition_check(X, J, eps[e])
            vav[12 * i + 4 * n_i + e] = sum(vi) / 10
    plt.plot(vav[12 * i: 12 * i + 4 * 1], marker='o')
    plt.plot(vav[12 * i + 4 * 1: 12 * i + 4 * 2], marker='o')
    plt.plot(vav[12 * i + 4 * 2: 12 * i + 4 * 3], marker='o')
    plt.legend([f'{m[i] * n_arr[0]}', f'{m[i] * n_arr[1]}', f'{m[i] *
n_arr[2]}'])
    plt.xlabel(f'epsilon = {eps[0], eps[1], eps[2], eps[3]}')
    plt.ylabel("V_av")
    plt.title(f'V_av plot for m = {m[i]}')
    plt.show()
    end_time = time.time()
    print(f'time: {end_time - start_time}')

for e in range(len(eps)):
```

```
    first_plot = [vav[e], vav[12 + e], vav[24 + e]]
    second_plot = [vav[4 + e], vav[16 + e], vav[28 + e]]
    third_plot = [vav[8 + e], vav[20 + e], vav[32 + e]]
    plt.plot(first_plot, marker='o')
    plt.plot(second_plot, marker='o')
    plt.plot(third_plot, marker='o')
    plt.legend([f'{m[i] * n_arr[0]}', f'{m[i] * n_arr[1]}', f'{m[i] *
n_arr[2]}'])
    plt.xlabel(f'm = {m[0], m[1], m[2]}')
    plt.ylabel("V_av")
    plt.title(f'V_av plot for epsilon = {eps[e]}')
    plt.show()
```