MAT160 - Final Project

Ahmed H. Mahmoud

June, 12th 2017

Problem No.1

Part A: Our code consists of first computing the left K singular vectors or rank K approximation $U_k^{(j)}$ (j represents the class) using the MATLAB function `svds` for each class $X^{(j)}$ using the training data set. For each test data point y_i, we compare the data point with all computed singular vectors (10 classes) and the inference value of the data point is the class of minimal error. The error of data point y_i w.r.t class j is computed as $E_j y_i = ||y_i - U_k^{(j)}(U_k^{(j)T} y_i)||_2$. Figure 1 shows the first five left singular vectors plotted as images for the first three classes (digits 0, 1 and 2). We notice that the first left singular vector (associated with largest singular value) closely resemble the corresponding digit followed by the second singular vector.

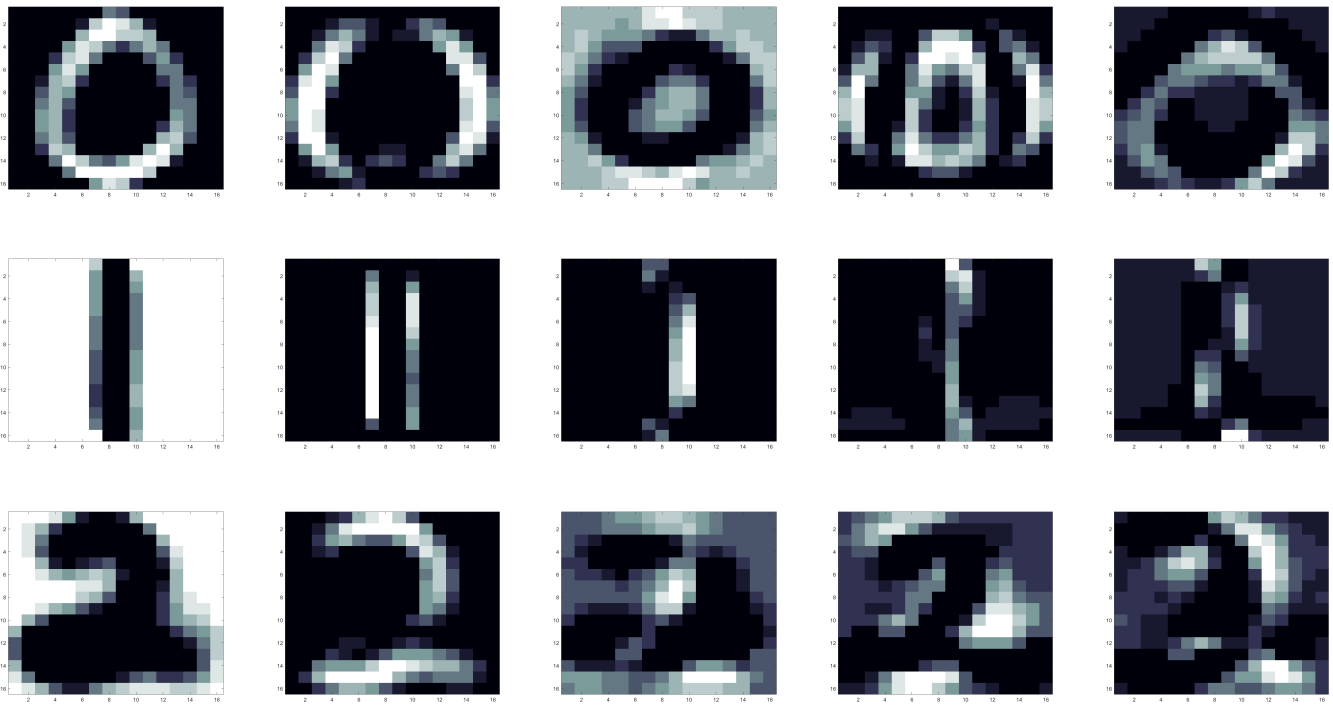
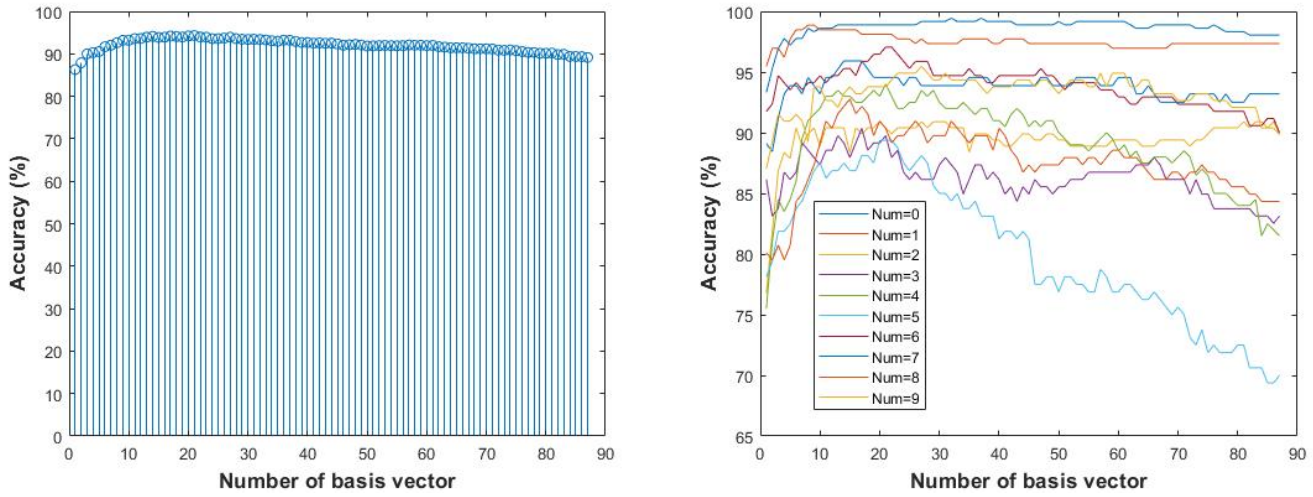


Figure 1: First left (associated with largest singular values) singular vectors for the first three classes.

Part B: We computed the accuracy as a function of the number of basis vector K. The accuracy is computed as the percentage of the number of images that has been classified correctly to all images that has been tested. Figure 2(a) shows the relation between accuracy and the number of basis vector. The maximum accuracy **94.32%** with **22** basis vector. The table to the right show the accuracy values for 5, 10, and 20 basis vector.

# Basis Vector	Accuracy
5	90.3%
10	93.2%
20	93.9 %



(a) (b)

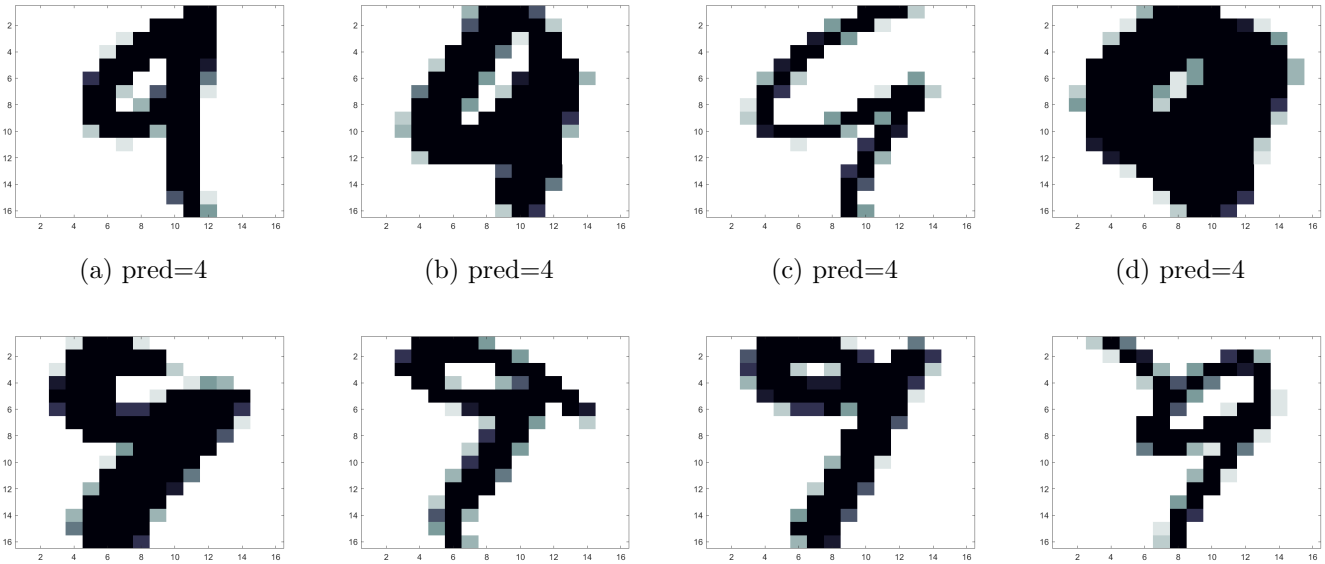
Figure 2: The accuracy of classified images as a function of the number of basis vectors K (a) for all images (b) per class/number.

Part C & D: In order to realize if it is beneficial to use different number of basis vector for different classes, we compute per-class accuracy for number of basis vector (from 2 up to 88). Figure 2(b) shows the per-class accuracy for different basis vector. We can see for class of *Number 5*, the accuracy decreases with increasing the number of basis vectors. For other classes like *Number 0* and *Number 1* the accuracy stays same (or marginally decreases) as the number of basis increases. For class of *Number 5*, it is better to use 21 vector basis for which the accuracy is 89.3%. Table 3 shows the maximum accuracy obtained for all class along with the number of basis vector associated with the maximum accuracy. We notice that class of *Number 1* only need 9 basis vector due to symmetry and simplicity of the shape of number one, while *Number 9* need up to 28 basis vector as it is not symmetric and the most complex.

In order to investigate the class of *Number 9* more thoroughly, we draw the few instances for which the number got mis-predicated using 28 basis vector as shown in Figure 4. Most of the mis-predicated images are indeed badly written and looks likes other number like the first row in Figure 4 where the number looks more like *Number 4* more than *Number 9*.

Class	Max Accuracy (%)	# Basis Vector
Number 0	99.4429	33
Number 1	98.8636	9
Number 2	90.9091	21
Number 3	90.3614	18
Number 4	94.0000	22
Number 5	89.3750	21
Number 6	97.0588	22
Number 7	95.9184	15
Number 8	92.7711	16
Number 9	95.4802	28

Figure 3: The maximum accuracy obtained per class by varying the number of basis vectors. For each class, we report here the maximum accuracy along with the number of basis vector associated with it.



(a) pred=4 (b) pred=4 (c) pred=4 (d) pred=4

(e) pred=8 (f) pred=8 (g) pred=7 (h) pred=7

Figure 4: Using 28 basis vector where the maximum accuracy is obtained for class of *Number 9*, we draw the few instances where the images is classified incorrectly along with the wrong predication

Problem No.2

Part A: Here we explain in details who to derive the matrix-form formulation. We have n date points, for each, we have $x_i \in \mathbb{R}^{11}$ features and $y_i \in \mathbb{R}$ label. The optimization problem is to find a and b such that

$$\underset{a\in\mathbb{R}^{11},b\in\mathbb{R}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} |a^T x_i + b - y_i|$$

We define $\omega \in \mathbb{R}^{12}$ as concatenation (in MATLAB language) of b on top of a. Similarly, we define

$\tilde{x} \in \mathbb{R}^{12}$ by concatenating 1 on top of x as follows

$$\omega \in \mathbb{R}^{12} = \begin{bmatrix} b \\ a_1 \\ a_2 \\ ... \\ ... \\ a_{10} \\ a_{11} \end{bmatrix} \qquad \tilde{x} \in \mathbb{R}^{12} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ ... \\ ... \\ x_{10} \\ x_{11} \end{bmatrix}$$

We can use ω and $\tilde{x}$ to redefine the optimization as follows

$$\underset{w \in \mathbb{R}^{12}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} |w^T \tilde{x}_i - y_i|$$

This can be written in matrix form by defining the matrix $X \in \mathbb{R}^{13 \times n}$ and vector $W \in \mathbb{R}^{13}$ as follows (superscript emphasizes different data points)

$$X \in \mathbb{R}^{13 \times n} = \begin{bmatrix} \tilde{x}_0^1 = 1 & \tilde{x}_0^2 = 1 & \cdots & \cdots & \cdots & \tilde{x}_0^n = 1 \\ \tilde{x}_1^1 = x_1^1 & \tilde{x}_1^2 = x_1^2 & \cdots & \cdots & \cdots & \tilde{x}_1^n = x_1^n \\ \tilde{x}_2^1 = x_2^1 & \tilde{x}_2^2 = x_2^2 & \cdots & \cdots & \cdots & \tilde{x}_2^n = x_2^n \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \tilde{x}_{11}^1 = x_{11}^1 & \tilde{x}_{11}^2 = x_{11}^2 & \cdots & \cdots & \cdots & \tilde{x}_{11}^n = x_{11}^n \\ -y^1 & -y^2 & \cdots & \cdots & \cdots & -y^n \end{bmatrix}, W \in \mathbb{R}^{13} = \begin{bmatrix} b \\ a_1 \\ a_2 \\ ... \\ ... \\ a_{10} \\ a_{11} \\ 1 \end{bmatrix}$$

Then the optimization problem can be re-written as

$$\underset{W \in \mathbb{R}^{13}}{\operatorname{argmin}} |X^T W|$$

Minimizing the 1-norm of a vector can be formulated as an Linear Programming (LP) problem as follows

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T S \\ \text{subject to} & X^T W \preceq S \\ & -X^T W \preceq S \end{array}$$

where $S \in \mathbb{R}^n$. From that, we can define the variables in the question to be: $c = \mathbf{1} \in \mathbb{R}^1$, $A = [X - X]^T \in \mathbb{R}^{2n \times 13}$, $d = S \in \mathbb{R}^{13}$, and $b = W \in \mathbb{R}^{13}$.

Part B: To solve the LP problem, we used `cvx`; MATLAB-based modeling system for convex optimization. `cvx` makes it easy to write and code the problem as a mathematical equations without much conversion compared with using `linprog` within MATLAB Optimization toolbox.

The optimal value we get was **0.49375**. The value represents the average absolute error of the linear model using the provided data set. Since the score (or labels) varies from 0 to 10, the values 0.49375 looks acceptable compared with the range of the scores.

The following shows the parameters of the model

$a = [\quad 0.083682 \quad -0.84129 \quad -0.23035 \quad 0.061643 \quad -1.608 \quad \cdots\cdots$

$\cdots \quad 0.0018835 \quad -0.0027255 \quad -61.517 \quad -0.041654 \quad 1.0888 \quad 0.29696]$

$b = 63.1391$

Part C: We used `cvx` to solve the least-squares regression problem by replacing the 1-norm by 2-norm. The model we get is as follows

$a = [\quad 0.025086 \quad -1.0835 \quad -0.18256 \quad 0.016374 \quad -1.8741 \quad \cdots\cdots$

$\cdots \quad 0.0043605 \quad -0.0032643 \quad -17.9835 \quad -0.41315 \quad 0.91647 \quad 0.2761]$

$b = 22.0655$

The residual sum of squared errors (RSS) is **666.4107**. Note that RSS $= \sum_{i=1}^{n}(y_i - a^T x_i - b)^2$.

Part D: Here we implemented the LASSO model by adding a regularization term to the least-squares regression model. The regularization is $\lambda|q|_1$ where $q \in \mathbb{R}^1 2 = [a; b]$ (i.e., extends the a vector to contain b as well). We experimented with few values for λ in order to reach a value that will turn most of the elements of q to zero except four of them; those that determines the quality of wine. With $\lambda = 0.2$, we found the four features are the top four: *volatile acidity*, *sulphates*, *pH*, and *alcohol*. Our model is as follows

$a = [\quad 0.063692 \quad -0.92564 \quad -5.3119 \times 10^{-5} \quad 1.2099 \times 10^{-5} \quad -8.1434 \times 10^{-6} \quad \cdots\cdots$

$\cdots \quad 0.0044334 \quad -0.0024063 \quad 0.00044866 \quad 0.38007 \quad 0.66368 \quad 0.32965]$

$b = 0.5012$

Note: In order to run the code, `cvx` should be installed on your machine.

Problem No.3

Part A: We construct a very simple mathematical model in order to solve the problem optimally. The model takes as input the edge list E. We have one decision variable $X \in \{0, 1\}$ such that $X_i = 1$ if vertex $i \in S$. The model then tries to directly maximize the number of vertices being influenced by maximizing the *sum* of vertices that are being influenced $N(S)$. Concretely, the model can be written as

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} max\left(X_i, \max_{(i,j)\in E} X_j\right) \\ \text{subject to} & |X|_1 \leq K \end{array}$$

Note that we choose to take the *max* since if we computed the *sum* instead, the computation won't reflect the problem correctly. This is because if a vertex i is connected to two vertices j and k such that $j \in S$ and $k \in S$, the sum will return 2 while we want it to return 1. However, we would expect that i returns just 1.

Even thought the model is really simple and straightforward, we could not execute it using SCIP and ZIMPL modeling language. This is primarily because ZIMPL does not allow *max* in the objective function. Another direction that we tried is to use an *if* statement. The model can be written as:

```
maximize myObjFunc : sum < V > in vertSet :
                    if (sum < V, j > in M : X[j] >= 0) then 1
                    else 0 end;
```

However, this model won't run because SCIP does not allow if statement in the objective function. To allow the model to run, we had to write it in away that only have *sum* expressions. The trick we used here is to express *max* and *min* using the Quadratic Formula. The quadratic formula defines $max(a, b) = \frac{1}{2}(a + b + |a - b|)$ and $min(a, b) = \frac{1}{2}(a + b - |a - b|)$. This formula only computes the *max* or *min* between two numbers. We can recognize that the (deep) inner *sum* in the objective function we defined at the beginning can be simply re-written as *min* of two numbers as follows

$$\max_{(i,j)\in E} X_j = \min(\sum_{(i,j)\in E} X_j, 1)$$

In other words, if the vertex i is not connected to any vertex that is in S, then this *min* will return 0 (min(0,1)). If vertex i is connected to one or more vertex that is in S, the *min* function will return 1. Since the *min* is between two numbers, we can use the Quadratic Formula and implement it using SCIP. The only thing left is to consider that $N(S)$ also include the vertices in S (the outer *max* in the objective function). Since this *max* is between two number, we can use the Quadratic Formula again. Finally, our objective function is expressed as

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} \max\left\{\min\left\{\sum_{(i,j)\in E} X_j, 1\right\}, X_i\right\} \\ \text{subject to} & |X|_1 \leq K \end{array}$$

where the *max* and *min* are written as in the Quadratic Formula. We implemented this model using SCIP. We had to pre-process the input graph such that and edge between i and j is written twice; one time as i the first vertex and second time as j. We also had to make sure that indexing starts with 1 (not 0). The model was tested it over small graph with 20 and 30 vertices with different values of K. At all cases, the model returns the optimal values correctly. However, we run the model using the facebook which has 4039 vertex and $K = 1$ (for which we know the optimal solution), the model keep running for a long time (3 hours) without giving an answer.

Practical method: Our practical method/algorithm is a greedy algorithm. We start with an empty set S. For number of iterations equal too the budget K, we add a new node v that maximize the *marginal gain*. The *marginal gain* is defined as the difference of the *influence* of $S \cup v$ the *influence* of S. The pseudocode of the algorithm is shown below

Algorithm 1 Influence Maximization Greedy Algorithm

1: **Input:** $G = (V, E)$
2: initialize $S = \emptyset$
3: **for** $i = 1$ to k **do**
4: select $u = argmax_{w \in V \setminus S} (I(S \cup \{w\}) - I(S))$
5: $S = S \cup \{u\}$
6: **end for**
7: **Output:** S

Guarantees: Let $I(S)$ be the the number of vertices influenced by set S returned by our greedy algorithm for some K, and $I(Opt)$ be the number of vertices influenced by the optimal set of vertices Opt of size K. Our algorithm is guaranteed to have the following bound

$$I(S) \geq (1 - 1/e) I(Opt)$$

The proof of this bound is derived from two property of the influence function I; 1) submodularity and 2) monotonicity.
Definition (Monotone): If S is a subset of T, then $I(S) \leq I(T)$ and $I(\emptyset) = 0$. This means that more people included in the set S, the more influence one will get (or at least the number of people influenced never decreased by increasing the size of S).
Definition (Submodular): If S is a subset of T, then for any node u we have

$$I(S \cup \{u\}) - I(S) \geq I(T \cup \{u\}) - I(T)$$

This means that adding a new vertex to the set (here T) has less impact ('marginal gain') than adding the same node to a smaller subset (here S) to that set. This is sometimes referred to as diminishing return.

Lemma 1. *If $B = \{b_1, \cdots, b_k\}$, then $I(A \cup B) - I(A) \leq \sum_{j=1}^{k} [I(A \cup \{b_j\}) - I(A)]$.*

Proof. Let B_i be the set containing the first i elements of B i.e., $\{b_1, \cdots, b_i\}$ (and $B_0 = \emptyset$). Now, we have k such sets, $B_1, B_2, \cdots, B_k$.
Now we can write $I(A \cup B) - I(A)$ as the following sum

$$(I(A \cup B_1) - I(A)) + (I(A \cup B_2) - I(A \cup B_1)) + \cdots + (I(A \cup B_k) - I(A \cup B_{k-1}))$$

Since I is submodular, $I(A \cup B_{i-1} \cup \{b_i\}) - I(A \cup B_{i-1}) \leq I(A \cup \{b_i\}) - I(A)$. From that, we have

$$\begin{aligned} I(A \cup B) - I(A) &= \sum_{i=1}^{k} [I(A \cup B_i) - I(A \cup B_{i-1})] \\ &= \sum_{i=1}^{k} [I(A \cup B_{i-1} \cup \{b_i\}) - f(A \cup B_{i-1})] \end{aligned}$$

$$\leq \sum_{i=1}^{k}[I(A \cup \{b_i\}) - I(A)]$$

■

Lemma 2. $\delta_{i+1} \geq (\frac{1}{k})(I(Opt) - I(S_i))$

Proof. δ_i is simply the marginal gain we get at step i i.e., $\delta_i = I(S_i) - I(S_{i-1})$. This lemma is concerned with setting a lower bound on the marginal gain we can get at each step. We can do this by summing up all the marginal gains. We also replace elements of the optimal solution with elements of the greedy solutions and see how that affect the marginal gain we get.
Suppose the optimal solution Opt is $\{t_1, t_2, \cdots, t_k\}$. Then,

$$\begin{aligned}
I(Opt) &\leq I(S_i \cup Opt) && \text{(monotonicity)} \\
&= I(S_i \cup Opt) - I(S_i) + I(S_i) \\
&\leq \sum_{j=1}^{k}[I(S_i \cup \{t_j\}) - I(S_i)] + I(S_i) && \textbf{(Lemma 1)} \\
&\leq \sum_{j=1}^{k}[I(S_{i+1}) - I(S_i)] + I(S_i)
\end{aligned}$$

This is because S_{i+1} is produced by choosing the element that maximizes the marginal gain. So we have $I(Opt) \leq \sum_{j=1}^{k} \delta_{i+1} + I(S_i) = I(S_i) + k\delta_{i+1}$. With a little rearrangement, we get $\delta_{i+1} \geq \frac{1}{k}[I(Opt) - I(S_i)]$. ■

Lemma 3. $I(S_{i+1}) \geq (1 - \frac{1}{k})I(S_i) + \frac{1}{k}I(Opt)$

Proof.

$$\begin{aligned}
I(S_{i+1}) &= I(S_i) + \delta_{i+1} \\
&\geq I(S_i) + \frac{1}{k}[I(Opt) - I(S_i)] \\
&\geq \left(1 - \frac{1}{k}\right) I(S_i) + \frac{1}{k}I(Opt)
\end{aligned}$$

■

Lemma 4. $I(S_i) \geq [1 - (1 - \frac{1}{k})^i I(Opt)], \forall i$

Proof. We can prove this by induction as follows
Base cases: For $i = 0$, $I(S_0) = I(\emptyset) = 0$, and the right hand side is also 0.
Inductive step: Assume the statement is true for S_i, and prove it is true for S_{i+1}. At $i + 1$, we have

$$\begin{aligned}
I(S_{i+1}) &\geq \left(1 - \frac{1}{k}\right) I(S_i) + \frac{1}{k}I(Opt) \\
&\geq \left(1 - \frac{1}{k}\right)\left(1 - \left(1 - \frac{1}{k}\right)^i\right) I(Opt) + \frac{1}{k}I(Opt) && \text{(by the induction hypothesis)} \\
&= [1 - \left(1 - \frac{1}{k}\right)^{i+1}]I(Opt)
\end{aligned}$$

■

Lemma 5. $I(S_k) \geq (1 - \frac{1}{e})I(Opt)$

Proof. From Lemma 4., $I(S) = I(S_k) \geq [1 - (1 - \frac{1}{k})^k]I(Opt)$.
We can use the inequality $1 + x \leq e^x$ (Bernoulli's inequality) by replacing $x = \frac{-1}{k}$. We get $\left(1 - \frac{1}{k}\right)^k \leq \left(e^{\frac{-1}{k}}\right)^k = \frac{1}{e}$. We then substitute to get

$$I(S) \geq \left(1 - \frac{1}{e}\right) I(Opt)$$

■

Part B: We implemented the greedy algorithm described above using MATLAB. The code read the edges of the graph from a text file and creates a `graph` object which is used to solve the problem. Shown below to instances of invoicing the code (stored in facebook.m).

Using $K = 1$

$>> facebook$

$Please\ \ enter\ \ the\ \ budget\ \ K\ \ (enter\ \ 0\ \ for\ \ K = |V|):\ \ 1$

$Step : 1$

$CurrentI(S) : 1045$

$The\ \ approximate\ \ optimal\ \ influence\ \ size\ \ (I(S)\ \ =\ \ |N(S)|) : 1046$

$The\ \ approximate\ \ optimal\ \ node\ \ set\ \ (S):\ \ 107$

$>>$

Using $K = 5$

$>> facebook$

$Please\ \ enter\ \ the\ \ budget\ \ K\ \ (enter\ \ 0\ \ for\ \ K = |V|):\ \ 5$

$Step : 1$

$Current\ \ I(S) : 1046$

$Step : 2$

$Current\ \ I(S) : 1823$

$Step : 3$

$Current\ \ I(S) : 2573$

$Step : 4$

$Current\ \ I(S) : 3120$

$Step : 5$

$Current\ \ I(S) : 3463$

$The\ \ approximate\ \ optimal\ \ influence\ \ size\ \ (I(S)\ \ =\ \ |N(S)|) : 3463$

$The\ \ approximate\ \ optimal\ \ node\ \ set\ \ (S) : 107$

1684

1912

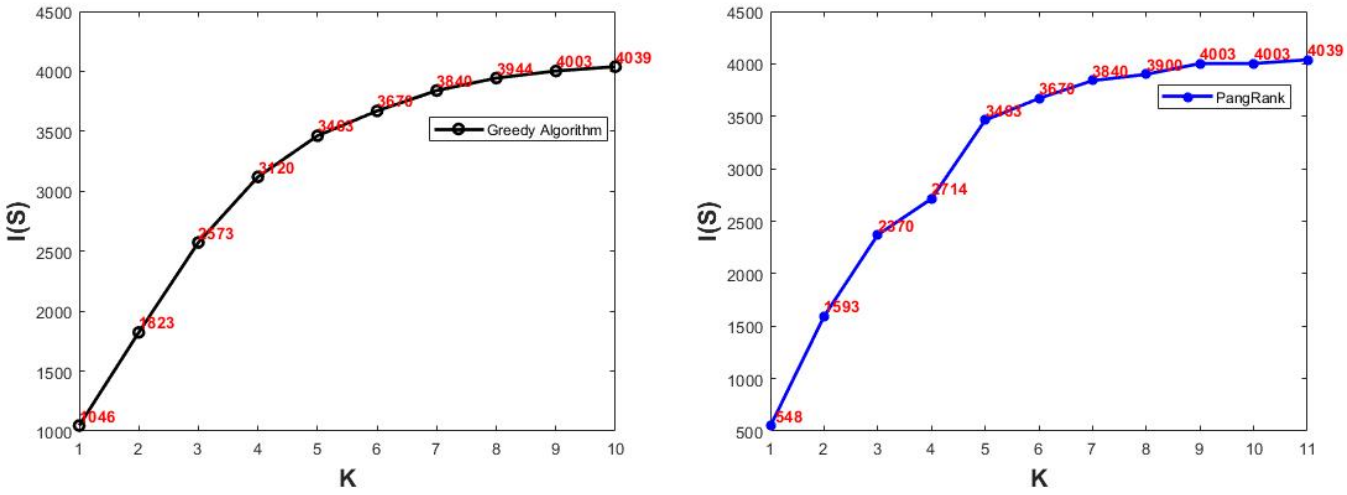
3437

0

>>

Part C: Figure 5(a) shows the relation between the budget K and the *influence* $I(S)$. After $K = 10$, the influence remains constant at 4039. This means that we only need to have ten vertices as a starting point to activate/influence all nodes in the graph. Note that if $v \in S$, then v is considered being influenced.

For $K = 1$, our greedy algorithm picks the node with largest number of neighbor nodes which happens to be node No. 107. As mention for $K > 10$, the whole graph will be influence i.e., for $K = |V|$, then $I(S) = |V|$. Our algorithm returns only 10 vertices for this case since, by definition, we are looking for $|S|$ to be at most K i.e., it is allowed to have $|S| \leq K$.



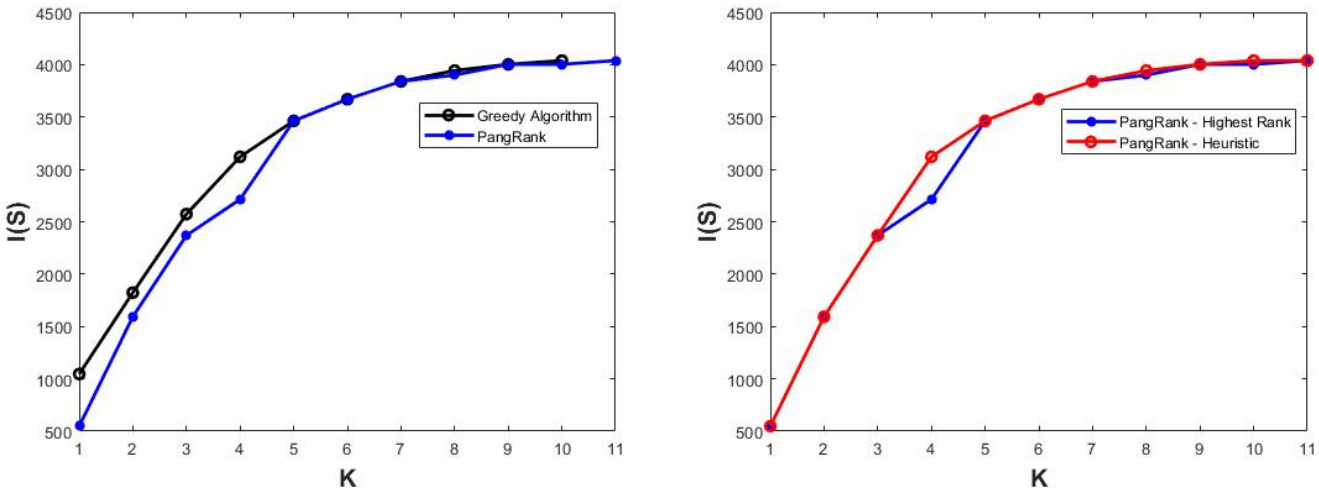
(a) Greedy Algorithm

(b) Pagerank Algorithm

Figure 5: The relation between the budget K and *influence* $I(S)$ using our greedy algorithm and pagerank implementation. Note that beyond $K = 10$ for the greedy algorithm (and $K = 11$ for pagerank algorithm), the *influence* remains constant at value of 4039 since after this value the whole graph would be influenced/activated.

Part D: We used the pagerank algorithm in order to identify the VIP nodes and compute S. We started by constructing the *column-stochastic matrix* P from which pagerank algorithm can compute the score for all vertices. Since pagerank algorithm is designed for directed graph and our graph undirected, we considered edge undirected edge as two undirected edges. From that we can easily construct P (as done in the first project). We then used pagerank to rank the different vertices. We tried two method to pick the VIP. The first one is simply to pick the vertices with highest pagerank score. This gave use almost the same vertices as the greedy algorithm gave. We were able to cover/influence the whole graph with $K = 11$. Figure 5(b). We superimposed both curved in Figure 6(a) to better visualize the performance of both. It is clear that the greedy algorithm outperform pagerank algorithm. However, with larger graphs, it is possible that pagerank might have better running time.

The second method we used to identify the VIP from the pagerank scoring is a heuristic. The method starts by adding the highest rank vertex to S. It then proceed by adding the vertex that will return highest marginal gain. We do this by looping over the vertices in descending order with respect to their score. At each step in the loop, we compute the marginal gain of this vertex and compare it with the marginal gain of the next vertices that have less score than this vertex and pick the vertex the has the highest marginal gain and add it to S. This heuristic method is a little superior than the first method as shown in Figure 6(b). We still need $K = 11$ in order to influence the whole graph. However, with $K = 4$, the heuristic return larger number of influenced vertices. It could be possible with larger graphs, there could be multiple values of K where heuristic method would be superior. Note that by design, the heuristic method could never give less influence that the first method; picking the vertices with highest score.



(a) Pagerank Algorithm (b) Pagerank Algorithm

Figure 6: Influence Maximization: (a) shows the comparison between the greedy algorithm performance and pagerank. (b) compares between two methods for picking the VIP vertices using pagerank algorithm; *Highest Rank* simply picks the K vertices with highest rank,*influence* applies some heuristic in order to imporve the performance a little.

Other methods to measure influence: One thing that our model(s) can't handle is the measuring the negative influence which can be added as negative weights over the edges.