4810-1183 Approximation and Online Algorithms with Applications
Lecture Note 5: Approximation Algorithm for Set Cover

Problem Definition

In this lecture note, we will consider the following optimization model:

Input: Set $V = \{1, \dots, n\}$, Set $E \subseteq 2^V$ (members of E is a subset of V)

Output: Set $S \subseteq V$

Constraint: For all $e \in E$, $S \cap e \neq \emptyset$

Objective Function: Minimize $|S|$

The optimization model is a generalization of the vertex cover problem discussed in the previous lecture note. In the vertex cover problem, each member of E is a set of 2 persons, while, in the set cover problem, the members can be a set of any size.

We introduce variables $x_i \in \{0,1\}$ for $i \in V$. When $i \in S$, we have $x_i = 1$, and we have $x_i = 0$ otherwise. For each $e = \{u_1, \dots, u_k\}$, we want to have one of $x_{u_1}, \dots, x_{u_k}$ be 1. That is

$$x_{u_1} + \cdots + x_{u_k} \geq 1.$$

Let **A** be a matrix with $|E|$ rows and $|V|$ columns, each column corresponds to a person in V and each row corresponds to a member of E. At each row corresponding to $e = \{x_{u_1}, \dots, x_{u_k}\}$, elements corresponding to $u_1, \dots, u_k$ are set to one, while the other are 0. Also, let $\mathbf{b}, \mathbf{c}$ be a vector, of which all elements are 1. We will have the following optimization model for the set cover problem:

Input: Matrix **A**, vectors $\mathbf{b}, \mathbf{c}$

Output: vector $\mathbf{x} = [x_1, \dots, x_n]^{\mathrm{t}}$, where $x_i \in \{0,1\}$

Constraint: $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$

Objective Function: Minimize $\mathbf{c}^{\mathrm{t}} \cdot \mathbf{x}$

Deterministic Rounding for Set Cover

The problem is NP-hard. However, because the problem is very close to the vertex cover problem, we may be able to solve the problem using deterministic rounding. We will again solve the following optimization model, called as "fractional set cover" by linear programming library:

Input: Matrix **A**, vectors $\mathbf{b}, \mathbf{c}$

Output: vector $\mathbf{x} = [x_1, \dots, x_n]^{\mathrm{t}}$, where $x_i \in [0,1]$

Constraint: $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$

Objective Function: Minimize $\mathbf{c}^{\mathrm{t}} \cdot \mathbf{x}$

Suppose that the optimal output from the library is $\mathbf{x} = [x_1, \dots x_n]^n$. In the previous lecture note, we output $\mathbf{x}' = [x_1', \dots, x_n']^t$ when $x_i' = 1$ for $x_i \geq 0.5$ and $x_i' = 0$ otherwise. Unfortunately, the output does not always satisfy the constraint. When $V = \{1,2,3\}$ and $E = \{1,2,3\}$, our only constraint is $x_1 + x_2 + x_3 \geq 1$. The optimal output from the library, $\mathbf{x}$, could be $\left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]^t$. Our output $\mathbf{x}'$ will be $[0,0,0]$. We have $S = \{\}$, which does not satisfy the constraint.

In the vertex cover problem, there are always only two variables at the left side of each constraint. In the set cover problem, we can have more. The value of 1 at the right side can be distributed to more variables. When we have 3 variables, the maximum value of the variables at the left side could be as small as $1/3$. When we have k, the value could be as small as $1/k$.

We change our rounding method based on the discussion on the previous paragraph. Let k be a size of the largest set in E, which is the largest number of variables on the left side. We will output $\mathbf{x}' = [x_1', \dots, x_n']^t$ when $x_i' = 1$ for $x_i \geq \frac{1}{k}$ and $x_i' = 0$ otherwise.

Theorem: The deterministic rounding mentioned in the previous paragraph is a k-approximation algorithm for the set cover problem.

Proof: Consider an arbitrary constraint. The constraint should have $m \leq k$ variables on the left side. Suppose that the constraint is

$$x_{u_1} + \cdots + x_{u_m} \geq 1.$$

We know that one of $x_{u_1}, \dots, x_{u_m}$ should be more than $\frac{1}{k}$. Otherwise, $x_{u_1} + \cdots + x_{u_m} \leq \frac{1}{k} + \cdots + \frac{1}{k} = \frac{m}{k} \leq 1$, and the constraint is not satisfied. Because of that, by the rounding mentioned, we will have one of $x_{u_1}', \dots, x_{u_m}'$ equal to 1. We will have $\mathbf{x}' = [x_1', \dots, x_n']^t$ that satisfies the constraint.

When OPT is the optimal value for a particular input, as $\mathbf{x}$ is an optimal solution from a minimization problem with more possible outputs (more flexible constraint), we have $\mathbf{c}^t\mathbf{x} \leq$ OPT. Also, because $x_i' \leq k \cdot x_i$, we have

$$SOL = \mathbf{c}^t\mathbf{x}' = \sum_i x_i' \leq k \cdot \sum_i x_i = k \cdot \mathbf{c}^t\mathbf{x} \leq k \cdot OPT.$$

■

We have k-approximation algorithm from the deterministic rounding. The approximation ratio k seems to be appealing in many applications, but not in many. In the worst case, $k = n$, and n-approximation algorithm is not appealing at all. A trivial algorithm that outputs all elements in V is also an n-approximation algorithm, because, for that case, $SOL = n$, $OPT \in [1, n]$, and $SOL \leq n \cdot OPT$.

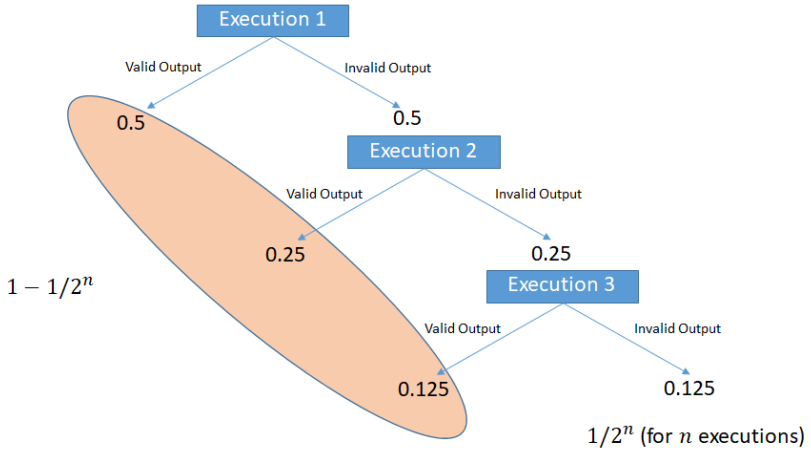
We observe that we assign too many variables to 1 in the deterministic rounding. For example, when we have a constraint $x_{u_1} + \cdots + x_{u_k} \geq 1$ and $x_{u_1} = \cdots = x_{u_k}$, we have $x_{u_1}' = \cdots = x_{u_k}'$ which means we select all the nodes $u_1, \dots, u_k$ to the set S where only one of them is required for this constraint. We should find a way that pick only a few of them, but, actually, it is not very easy to pick. Randomness sometimes can help us passing through the complication.

Randomized Algorithm

Randomized algorithms are algorithms that will work differently on each execution. However, the different work will be controlled by a probability distribution. For example, we might set `a = 0` with probability 0.7, and set `a = 1` with probability 0.3. During the program execution, we will toss a (biased) coin and assign the value of a based on the coin value.

Because our program will work differently on each execution, we might get an optimal solution at some execution. On the other hand, we may get an output with a very bad objective value, or even an output that does not satisfy our constraint. However, there is a way to prevent that. Suppose that, at each iteration, we have an output that satisfies the constraint with probability 0.5. We have not got the desirable output there with probability 0.5, but, for that case, we can rerun the algorithm again. The success probability after the second iteration is $0.5 + 0.5 \times 0.5 = 0.75$. The following figure will

show that the success probability is $1 - 1/2^n$ after the n^{th} iteration. When n is as large as 30, the failure probability is $1/2^{30}$, which is much smaller than the hardware failure probability.



Randomized Rounding

We will introduce the concept of randomized algorithm to rounding. Previously, we have $x_i' = 1$ when x_i is large enough. Now, we will have $x_i' = 1$ with a larger probability when x_i is large. To be more precise, we will have $x_i' = 1$ with probability x_i.

For example, when $k = 3$ and we have a constraint $x_1 + x_2 + x_3 \geq 1$, suppose that we have $x_1 = 0.6, x_2 = 0.4, x_3 = 0$ from the linear programming library. In deterministic rounding, we will have $x_1' = x_2' = 1$ and $x_3' = 0$, but, in randomized rounding, we will have $x_1' = 1$ with probability 0.6, $x_2' = 1$ with probability 0.4, and $x_3' = 1$ with probability 0.

Let try to calculate the probability that the constraint is satisfied. On this example, the constraint will not satisfy if all x_1', x_2', x_3' are 0. The probability is $(1 - 0.6) \cdot (1 - 0.4) \cdot 1 = 0.24$. For general x_1, x_2, x_3, the probability is

$$(1 - x_1) \cdot (1 - x_2) \cdot (1 - x_3) \leq e^{-x_1} \cdot e^{-x_2} \cdot e^{-x_3} = e^{-(x_1 + x_2 + x_3)}.$$

Because we always have $x_1 + x_2 + x_3 \geq 1$, we have $e^{-(x_1 + x_2 + x_3)} \leq e^{-1} \leq 0.37$. The probability that the constraint is satisfied is about 0.63.

We can apply the calculation in the previous paragraph to any other constraint. We will know that the probability that a constraint is satisfied is more than 0.63. That probability looks promising. However, because we want to satisfy <u>all</u> constraints, when the number of constraints is 100, having each constraint failed with probability 0.37 is not good enough.

We now know that having $x_i' = 1$ with probability x_i is not so nice. The probability might be too small. Let us try to have a larger probability. Let us try a probability $1 - (1 - x_i)^t$ for some integer $t > 1$. The failing probability for the constraint $x_1 + x_2 + x_3 \geq 1$ would be

$$\begin{aligned}&\left(1 - (1 - (1 - x_1)^t)\right) \cdot \left(1 - (1 - (1 - x_2)^t)\right) \cdot \left(1 - (1 - (1 - x_3)^t)\right) \\ &\qquad = (1 - x_1)^t \cdot (1 - x_2)^t \cdot (1 - x_3)^t \leq e^{-x_1 t} e^{-x_2 t} e^{-x_3 t} = e^{-t \cdot (x_1 + x_2 + x_3)} \leq e^{-t}.\end{aligned}$$

Again, the calculation can be apply to any constraint, so the failing probability for each constraint is e^t. Now, let calculate a probability of having some constraint failed. Because the number of constraint is $|E|$, we have

$$\Pr[\text{some constraint failed}] \leq \sum_C [\text{constraint } C \text{ failed}] \leq e^{-t} \cdot |E|.$$

When the failing probability of an execution is no more than $1/2$, we can iterate the executions many times to have an arbitrary small failing probability. Because of that, we target to have $\Pr[\text{some constraint failed}] \leq 1/2$. To guarantee that, we need $e^{-t} \cdot |E| \leq 0.5$.

$$e^{-t} \leq 0.5/|E|$$

$$\ln e^{-t} \leq \ln \frac{1}{2|E|}$$

$$-t \leq -\ln 2|E|$$

$$t \geq \ln 2|E| = \ln|E| + \ln 2.$$

The probability that we have $x_i' = 1$ should be equal to $1 - (1 - x_i)^{\ln 2|E|}$.

The randomized rounding algorithm is summarized as follows:

1: Use a linear programming library to solve the fractional set cover problem. Suppose that the output is $\mathbf{x} = [x_1, \ldots, x_n]^t$.

2: Let $\mathbf{x}' = [x_1', \ldots, x_n']^t$ where $x_i' = 1$ with probability $1 - (1 - x_i)^{\ln|E| + \ln 2}$.

3: If $\mathbf{x}'$ satisfies the constraint outputs $\mathbf{x}$, otherwise goto 1

We will analyze the objective value of the randomized rounding algorithm from now. Again, we will denote the objective value of the output from our algorithm by SOL, and denote the optimal value by OPT. In the following theorem, we will use Markov inequality. For any random variable of which the expected value is EXP, the probability that the variable value is more than a is no more than EXP/a. By that, the probability that the variable is more than $2 \cdot E$ is no more than $\frac{EXP}{2 \cdot EXP} = 1/2$.

Theorem: With probability $1/2$, $SOL \leq 2 \cdot (\ln|E| + \ln 2) \cdot OPT$.

Proof: By the notations, we have $SOL = \sum_i x_i'$. Each person i will contribute 1 to $\sum_i x_i'$ with probability $1 - (1 - x_i)^{\ln|E| + \ln 2}$ and contribute 0 with probability $(1 - x_i)^{\ln|E| + \ln 2}$. The expected contribution (expected value of x_i') is then

$$1 \cdot \left[1 - (1 - x_i)^{\ln|E| + \ln 2}\right] + 0 \cdot (1 - x_i)^{\ln|E| + \ln 2} = 1 - (1 - x_i)^{\ln|E| + \ln 2}.$$

Because of that, the expected value of $\sum_i x_i'$ is $\sum_i [1 - (1 - x_i)^{\ln|E| + \ln 2}]$. Because $1 - (1 - x_i)^{\ln|E| + \ln 2} \leq (\ln|E| + \ln 2)x_i$, we have

$$\text{Expected value of } SOL = \sum_i [1 - (1 - x_i)^{\ln|E| + \ln 2}] \leq \sum_i [(\ln|E| + \ln 2)x_i] = (\ln|E| + \ln 2) \sum_i x_i$$
$$\leq (\ln|E| + \ln 2)OPT.$$

By the Markov inequality, the probability that the SOL is more than $2 \cdot EXP = 2 \cdot (\ln|E| + \ln 2) \cdot OPT$ is no more than 0.5.

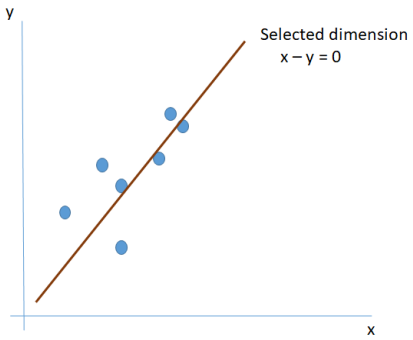
■

By randomized rounding, we reduce the approximation ratio from k to around $2 \cdot (\ln|E| + \ln 2) \leq 4 \ln n + O(1)$.

Principle Component Analysis (PCA)

PCA is one of the most conventional machine learning technique for dimensionality reduction. Suppose that we are considering a data with large dimension. It is usually very hard to extract information using any machine learning algorithm in this situation. However, in many cases, it is possible to represent the data using few dimensions without losing a lot of information.

Several works propose how to select those few dimensions. In PCA, the dimensions selected are not original dimensions, but a "linear combination" of dimensions. In the following figure, the selected dimension is not "*x*" or "*y*" but the brown line, which can be denoted by the equation $x - y = 0$. We can also denote the dimension by a vector of which elements are coordinate in the equation. In the example, we can denote the dimension by vector $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$.



Suppose that we have n records and d dimensions and all data are real numbers. We can denote all the data by matrix **X** with n rows and d columns. Each row corresponds to a record, and each column corresponds to a dimension. PCA claims that a combination of dimensions, denoted by **v**, can represent the maximum amount of information in **X**, if it is a unit vector that maximize $\mathbf{v}^t\mathbf{X}^t\mathbf{X}\mathbf{v}$. We have the following optimization model:

Input: Matrix **X**

Output: Vector $\mathbf{v} = [v_1, \ldots, v_n]^{\mathrm{t}}$

Constraint: $v_1^2 + \cdots + v_n^2 = 1$ (**v** is a unit vector.)

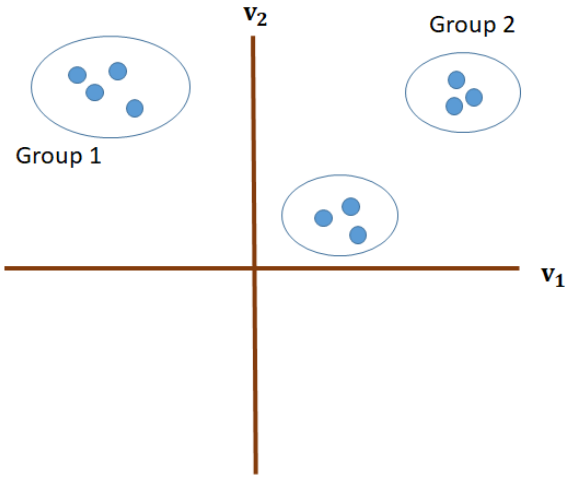
Objective Function: Maximize $\mathbf{v}^t\mathbf{X}^t\mathbf{X}\mathbf{v}$

Fortunately, it is possible to solve the above optimization model. That could be a reason why PCA is one of the most well-known dimensionality reduction technique.

Sparse PCA

Consider the case when we have a data that contains frequencies of words in all books at a library. Each record corresponds to a book and each dimension corresponds to a word. We want to cluster the books into groups. As the number of words is as large as 40,000, we want to use a dimensionality reduction algorithm. Suppose that we use PCA, and have 2 dimensions $\mathbf{v_1}, \mathbf{v_2}$ from the algorithm. We cluster the books based on $\mathbf{v_1}, \mathbf{v_2}$. An example of results is shown in the figure in the following page.

After we get the clustering results, we want to know what books in Group 2 is about. We know from the results that the values in dimensions $\mathbf{v_1}$ and $\mathbf{v_2}$ are large. However, as $\mathbf{v_1}$ and $\mathbf{v_2}$ are combinations of large number of dimensions (words), it is quite hard to make sense of the clustering results.



If the dimensions $\mathbf{v_1}, \mathbf{v_2}$ are not combinations of all words, but combinations of only few words. We might be able to check what those few words are about, and guess what groups stand for. Because of that, we would like to limit the number of non-zero elements in $\mathbf{v_1}$ and $\mathbf{v_2}$. We then have the following optimization model.

Input: Matrix $\mathbf{X}$, integer k

Output: Vector $\mathbf{v} = [v_1, \dots, v_n]^{\mathrm{t}}$

Constraint: $v_1^2 + \cdots + v_n^2 = 1$ ($\mathbf{v}$ is a unit vector.)
Number of non-zeros in $\mathbf{v}$ is no more than k.

Objective Function: Maximize $\mathbf{v}^t\mathbf{X}^t\mathbf{X}\mathbf{v}$

We call the above optimization model as "sparse PCA". The optimization model is unfortunately NP-Hard, and we need approximation algorithm for the problem.

Randomized Rounding for Sparse PCA

In this section, we will discuss an algorithm for sparse PCA proposed in [2]. The algorithm first solve the following optimization model.

Input: Matrix $\mathbf{X}$, integer k

Output: Vector $\mathbf{v} = [v_1, \dots, v_n]^{\mathrm{t}}$

Constraint: $v_1^2 + \cdots + v_n^2 = 1$ ($\mathbf{v}$ is a unit vector.)
$|v_1| + \cdots + |v_n| \leq \sqrt{k}$.

Objective Function: Maximize $\mathbf{v}^t\mathbf{X}^t\mathbf{X}\mathbf{v}$

In word, instead of having small number of non-zeros (small ℓ_0), we want to have a small sum (small ℓ_1). It is known that, while it is usually very hard to deal with ℓ_0, it usually much easier to deal with ℓ_1. Actually, we can solve the ℓ_1 problem using libraries for semi-definite programming (SDP).

The problem here is how to converse a vector with small ℓ_1 to a vector with small ℓ_0. As you may guess, we will use the randomized rounding technique. Suppose that we have $\mathbf{v} = [v_1, \dots, v_n]^t$ from the library for ℓ_1 problem. From $\mathbf{v}$, we want to find a solution for the sparse PCA problem, denoted by $\mathbf{v}' = [v_1', \dots, v_n']$. For a positive integer s that we will define later, we have

$$p_i := \min\left\{s \cdot \frac{|v_i|}{|v_1| + \cdots + |v_n|}, 1\right\}.$$

We will have $v_i' = v_i/p_i$ with probability p_i, and $v_i' = 0$ otherwise. When $|v_i|$ is large, the probability that we have $v_i' \neq 0$ is also large. On the other hand, when $|v_i|$ is small, we are likely to have $v_i' = 0$.

The expected number of non-zeros in v′ is $\sum_i p_i$. Because $p_i \leq s \cdot \frac{|v_i|}{|v_1|+\cdots+|v_n|}$, we know that the expected number is no more than

$$\sum_i s \cdot \frac{|v_i|}{|v_1| + \cdots + |v_n|} = \frac{s}{|v_1| + \cdots + |v_n|} \sum_i |v_i| = s.$$

When $s = k$, the expected number of non-zeros is k.

We have the constraint on number of non-zeros satisfied. Let consider the constraint $\sum_i (v_i')^2 = 1$. We know that $(v_i')^2 = v_i^2/p_i^2$ with probability p_i and equal to 0 with probability $1 - p_i$. The expected contribution of $(v_i')^2$ to the summation is:

$$p_i \cdot \frac{v_i^2}{p_i^2} + (1 - p_i) \cdot 0 = \frac{v_i^2}{p_i}.$$

We will separately consider the case when $p_i = 1$ and $p_i = s \cdot \frac{|v_i|}{|v_1|+\cdots+|v_n|}$. When $p_i = 1$, the contribution is v_i^2. When $p_i = s \cdot \frac{|v_i|}{|v_1|+\cdots+|v_n|}$, the contribution is

$$\frac{v_i^2}{p_i} = \left(\sum_i |v_i|\right) \cdot \frac{v_i^2}{s \cdot |v_i|} = \frac{|v_i|}{s} \cdot \sum_i |v_i| \leq \frac{|v_i|}{s}\left(|v_i| + \sum_i |v_i|\right) \leq \frac{v_i^2}{s} + \frac{|v_i|}{s} \cdot \sum_i |v_i|$$
$$\leq v_i^2 + \frac{|v_i|}{s} \cdot \sqrt{k}.$$

When we sum all the contributions together, we have

$$\sum_{p_i=1} v_i^2 + \sum_{p_i<1} \left(v_i^2 + \frac{|v_i|}{s} \cdot \sqrt{k}\right) = \sum_{p_i=1} v_i^2 + \sum_{p_i<1} v_i^2 + \sum_{p_i<1} \frac{|v_i|}{s} \cdot \sqrt{k} = \sum_i v_i^2 + \frac{\sqrt{k}}{s} \sum_i |v_i|$$
$$= 1 + \frac{\sqrt{k}}{s} \cdot \sqrt{k} = 1 + \frac{k}{s}.$$

Although we want to have $\sum_i v_i^2 = 1$, we unfortunately have the expected value of $\sum_i v_i^2$ equal to $1 + \frac{k}{s}$. When $s = k$, we will have the expected value no more than 2. That violates the constraints of the sparse PCA problem.

We now have a dilemma between two constraint. Recall that the expected number of non-zeros is s, while the expected value of $\sum_i v_i^2$ is $1 + \frac{k}{s}$. We will have a lot of non-zeros when s is large, while we will have a large $\sum_i v_i^2$ when s is small. Experimentally, the authors have not got a nice result when s is as large as $200 \cdot k$. The expected value of $\sum_i v_i^2$ will be no more than $1 + \frac{k}{200k} = 1.005$ (which is very close to 1. The number of non-zeros is can be as large as $200 \cdot k$, but, in the experimental results, the authors have a much smaller number.

The randomized rounding algorithm is then as follows:

1: Solve the ℓ_1-sparse PCA problem using SDP library.
Suppose that the output from the library is $\mathbf{v} = [v_1, \dots, v_n]^t$.

2: Output $v' = [v'_1, \dots, v'_n]$ when $v'_i = v_i / p_i$ with probability $p_i \coloneqq \min\left\{200k \cdot \frac{|v_i|}{|v_1|+\cdots+|v_n|}, 1\right\}$, and 0 otherwise.

Suppose that OPT is the optimal value and SOL is the objective value of the solution from the above algorithm. Because we are working on the maximization problem, $SOL \leq OPT$. The author can prove the following theorem:

Theorem: For any particular input, $SOL \geq OPT - 1$.

As the proof contains a lot of calculation, we will skip the proof in this lecture note.

Previously, we usually have $SOL \geq \alpha \cdot OPT$, while α is called as approximation ratio. We have $SOL \geq OPT - \beta$ in the above theorem. The variable β is usually called *additive approximation ratio*.

The fact that the difference between SOL and OPT is no more than 1 looks promising. However, when OPT is as small as 0.5, $OPT - 1 = -0.5$. We can only guarantee $SOL \geq -0.5$, which is trivial because any vector $\mathbf{v}$ will have $\mathbf{v}^t\mathbf{X}^t\mathbf{X}\mathbf{v} \geq 0$. This is the reason why the additive approximation ratio is not as common as the multiplicative approximation ratio – it cannot provide a good guarantee for any input.

Actually, the authors can prove that $SOL \geq OPT - \epsilon$ for $s = \frac{200k}{\epsilon^2}$.

References

[1] D. P. Williamson and D. Shmoys, "*The design of approximation algorithms*", Cambridge University Press, 2011. Chapter 5

[2] K. Fountoulakis, A. Kundu, E. Kontopoulou, and P. Drineas, "A randomized rounding algorithm for sparse PCA", ACM Transactions on Knowledge Discovery from Data (TKDD), Vol. 11, No. 3, 2017.