15.095 Machine Learning Under Modern Optimization Lens

Solving the Learning Parity with Noise Problem using ML and MIO

Team Members: Arushi Jain (jarush08@mit.edu), Jan Reig Torra (janreig@mit.edu)

INTRODUCTION AND MOTIVATION

Imagine that Alice and Bob are two agents on a secret mission. Prior to every communication between them, they identify each other with the help of a shared secret key *s*. If Alice wants to identify herself to Bob, she needs to share her secret key with Bob so that Bob can verify that she is indeed Alice and not an imposter. However, by directly communicating the secret key *s*, an adversary may sniff the key and can later re-use it. Hence, this method of direct communication is not safe.

To tackle this, Bob can instead ask Alice a set of questions that Alice can answer using her secret key. The questions can ask Alice to reveal some bits at a time but not all of them at once. The questions asked by Bob and the answers given by Alice can still be overheard by the adversary, however, now the adversary will have to reverse engineer the key from the set of questions and answers. It is difficult but still tractable since the key can be recovered by solving a system of linear equations if the set of questions and answers is known.

However, Alice and Bob can still make the life of the adversary more difficult by adding slight random noise to the answers instead of providing the exact answers. The adversary will no longer be able to solve a linear system and the possibilities of the secret key will be exponential just by adding random noise. This is called Learning Parity with Noise (LPN) Problem.

PROBLEM STATEMENT

Imagine you are the adversary. Solve the LPN Problem for an n-dimensional secret key $s \in \{0,1\}^n$ over m questions given by question matrix $A \in \{0,1\}^{m*n}$ and m answers given answer vector $b \in \{0,1\}^m$. The answers can have random noise $e \in \{0,1\}^m$ that follows a Bernoulli distribution with noise parameter $p < 0.5$. The relationship between the questions, answers, noise, and secret key is given by:

$$A * s + e = b$$

Intuition

Here, each row in matrix A represents a question about revealing specific bits of the secret key s. For example, over a 4-dimensional key s = [1 1 0 1], a question that asks about revealing the sum of 2nd and 4th bits will be represented by the vector [0 1 0 1]. The answer will be given by the XOR multiplication of matrix A with secret key s:

$$[0\ 1\ 0\ 1] * [1\ 1\ 0\ 1]^T = 0*1 + 1*1 + 0*0 + 1*1 = 0 + 1 + 0 + 1 = 0$$

Hence, our answer is 0 which is represented by the corresponding row of vector b. We will have m such questions and answers. This process of revealing some but not all bits at a time is called HB Protocol[1].

Now, *e* is our noise vector that follows Bernoulli's distribution with probability p. This means that we flip a biased coin that returns heads (or 1) with probability *p* and tails (or 0) with probability *1-p*. In our previous example, if the corresponding element of the error vector is 0, we get our final answer as 0 + 0 = 0 (which corresponds to the correct sum). However, if the corresponding element of the error vector is 1, our final answer is 0 + 1 = 1 (which corresponds to the incorrect sum). Since e is an m-dimensional vector, it will have approximately $p * m$ elements as 1.

We want to recover the secret key s, given Bob's input questions and Alice's noisy output answers.

[1] Nicholas J. Hopper and Manuel Blum. Secure human identification protocols. ASIACRYPT 2001

LITERATURE SURVEY

There has been extensive work on the LPN problem over the decades and the problem is known to be NP-Hard[2]. The best key recovery attacks are exponential in time. There are two classes of algorithms that solve the LPN Problem – 1) Combinatorial algorithms such as the BKW algorithm[3] which is comparatively independent of the Bernoulli noise p and is the fastest algorithm for constant threshold, making it a good choice to attack HB Protocol and its variants, 2) Decoding algorithms such as Prange's information set decoding[4] which has fully exponential run-time but outperforms BKW on low-noise LPN problems.

ML algorithms and MIO have not been used in the past to solve this problem to the best of our knowledge.

DATASET

We will generate synthetic data for our problem using a fixed error probability ($p < 0.5$) and a secret key (s) of length n. Based on n and p, we will construct the dataset by simulating the question matrix A with dimensions m x n where m is given by[5]:

$$m \leq 4 * n * (0.5 - p)^{-2} \quad (A)$$

We will also simulate a noise vector e that follows a Bernoulli distribution with error probability p. Finally, we will generate our answer vector using the following equation:

$$(A * s + e) \,\%\, 2 = b \quad (B)$$

Here, we compute modulo 2 since $A * s$ is XOR product and $A * s + e$ is XOR addition.

SOLUTION TECHNIQUES

1. Machine Learning

Given a fixed p, we will train an ML model for a synthetic input matrix A and output vector b in order to recover the secret key s. Note that this problem is a classification problem since each element of b is either 0 or 1.

We will try various classification techniques such as Logistic Regression, SVM, Optimal Feature Selection, CART, OCT, OCT-H, Random Forest Classifier, and XGBoost Classifier. We will perform 3-fold cross-validation for hyperparameter tuning for sparsity, minbucket, depth, number of trees, and estimators.

Evaluation

From our trained model, we will make predictions on test set I^{n*n}. We use the identity matrix of dimension n as our test set to recover each bit of the predicted secret key $\hat{s}$. From our previous example, the multiplication of a 4x4 identity matrix with secret key s, gives the secret key itself. Hence, we should get predictions for the identity set on the trained ML model.

To check our secret key $\hat{s}$, we will compute how much our predicted answer differs from the true answer:

$$sum\big((A * \hat{s} + b) \,\%\, 2\big) = error \quad (C)$$

Note that $(A * \hat{s} + b) \,\%\, 2$ returns 1 if $A * \hat{s}$ is different from b, otherwise it returns 0.

We will also compute an error threshold to check if our error is small enough:

---

2 Hopper, N.J., Blum, M. Secure Human Identification Protocols. In: Boyd, C. (eds) Advances in Cryptology - ASIACRYPT 2001

3 Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model, 32nd ACM STOC

4 Eugene Prange. The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory

5 R. Kübler. Time-Memory Trade-Offs for the Learning Parity with Noise Problem (2018), Dissertation (Ruhr University Bochum)

$$\tau = p * m + \delta \qquad (D)$$

We will assume that the key is recovered if $error \leq \tau$. Note that we need to do these gymnastics to evaluate our model since we don't have the real key from Alice but only her noisy answers.

2. Mixed Integer Optimization

Given a fixed p, we will solve an MIO problem for a synthetic input matrix A and output vector b in order to recover the secret key s. Note that this problem will be a mixed-integer problem since each element of key s is either 0 or 1 and b is the XOR product of A and s plus some error term. The advantage of this approach will be that it will always recover the key for reasonable number of samples, unlike the ML models.

Our objective will be to minimize the difference between $A * s$ and b since this difference corresponds to our Bernoulli error p:

$$\min_{s \in \{0,1\}^n} \quad \sum_{j=1}^{m} |a_j * s_j - b_j|$$

Now since everything is a XOR operation here, we need to model the constraints of XOR multiplication and XOR addition. To be able to do that, for every multiplication and addition operation, we need to compute the result of the operation modulus 2. Let h_j be the remainder when $a_j * s_j$ is divided by 2. Our objective now becomes:

$$\min_{s,h,y} \sum_{j=1}^{m} |h_j - b_j|$$

$$s.t. \quad \sum_{i=1}^{n} A_{ji} * s_i = 2y_j + h_j \qquad j \in m$$

$$\boldsymbol{h}, \boldsymbol{y} \geq 0$$

$$h_j < 2 \qquad j \in m$$

$$\boldsymbol{s} \in \{0,1\}^n, \boldsymbol{h} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{Z}^m$$

We're almost there, now let's linearize it and get the final MIO formulation!

MIO Formulation

$$\min_{s,h,z,y} \sum_{j=1}^{m} z_j$$

$$s.t. \quad \sum_{i=1}^{n} A_{ji} * s_i = 2y_j + h_j \qquad j \in m \qquad (1)$$

$$h_j - b_j \leq z_j \qquad j \in m \qquad (2)$$

$$-h_j + b_j \leq z_j \qquad j \in m \qquad (3)$$

$$h_j < 2 \qquad j \in m \qquad (4)$$

$$\boldsymbol{h}, \boldsymbol{y}, \boldsymbol{z} \geq 0 \qquad (5)$$

$$\boldsymbol{s} \in \{0,1\}^n, \quad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^m, \quad \boldsymbol{y} \in \mathbb{Z}^m \qquad (6)$$

- Constraint (1) finds the XOR product of A and s. To achieve this, we are dividing the product of A and s by 2 to calculate the remainder h which is forced to be less than 2 in constraint (4). Note

that according to constraint (6), the quotient y is constrained to be an integer and secret key s is constrained to be a binary

- Since we want to minimize the absolute difference between h and b, we linearize this objective by introducing a new variable z and new constraints (2) and (3)
- Our linear objective is thus, to minimize the sum over z

3. Brute Force Algorithm

We will also use the Brute-Force algorithm to find the secret key s by iterating over all possible keys until a given key satisfies the threshold criteria in (D). The solution from this approach will serve as a benchmark both in terms of correctness as well as time complexity.

Algorithm: Brute Force
Input: A, b, τ **Output:** s

n, m = dim(A)
for $s \in \mathbb{F}_2^n$
  if $sum(A * s + b) \leq \tau$ **then**
  return s

EXPERIMENT SETUP

We will test our ML solution and MIO solution over a range of values of n, m and s with different combination ratios of 0 and 1 and analyze in which cases we're able to recover the secret key and with how much computational time.

Machine Learning

Aim: To understand if ML models recover the key correctly

For each n from 4, 6….32, we will train ML models to recover the n-dimensional secret key *s* correctly. We will run multiple experiments as explained below:

- Generating 5 combinations of keys with 0%, 12.5%, 25%, 37.5% and 50% ratio of 1 vs 0 for each key length
- Training the ML models 25 times on m different inputs and outputs to see how many times each model is able to recover the key correctly where m is computed from equation (A)
- Running it over 8 ML models: GLMNet, SVM, Optimal Feature Selection (OFS), CART, Random Forest (RF), XGBoost (XGB), OCT, OCT-H with hyperparameter tuning using 5-fold cross-validation

Total Experiments = $15 * 5 * 25 * 8 = 15000$

Mixed Integer Optimization and Brute Force

Aim:

i. To understand if MIO needs less examples than Brute Force to recover the key
ii. To compute how slow is the MIO approach compared to Brute Force

For each n from 4, 6….20, we will run the Brute Force Algorithm and the MIO to recover the n-dimensional secret key *s* correctly. We will run multiple experiments as explained below:

- Generating 5 combinations of keys with 0%, 25%, 50%, 75% and 100% ratio of 1 vs 0 for each key length
- Recovering the key over ~50 values of m from Brute Force and MIO on different inputs and outputs where the 5 values of m are $\{n, n + \frac{M}{50}, n + \frac{2M}{50}, n + \frac{3M}{50}, \dots M\}$, M is given by equation (A)

Approximate Total Experiments = $2 * 9 * 5 * 50 = 4500$

RESULTS AND INFERENCES

Machine Learning

Table 1 shows the proportion of times we correctly recover the key (out of 25) for various ML models for ratios 0.25 and 0.50. We are not showing the results for ratio = 0, 0.125 since the key is recovered in most cases by all models except GLM, OFS and SVM. We are also omitting results for ratio = 0.375 in the interest of space but the performance lies between that of ratio = 0.25 and 0.5 for all ML models.

N	Ratio	Key Recovery Success Rate (25 Trials)							
		OCT	OCT-H	CART	RF	XGB	GLM	OFS	SVM
4	0.25	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	0.50	1.00	1.00	0.84	1.00	0.96	0.20	0.28	0.28
6	0.25	1.00	1.00	1.00	1.00	0.92	1.00	1.00	1.00
	0.50	1.00	1.00	0.52	0.44	0.68	0.12	0.16	0.16
8	0.25	1.00	1.00	0.52	0.52	0.80	0.12	0.28	0.28
	0.50	0.80	0.92	0.40	0.28	0.48	0.00	0.08	0.08
10	0.25	1.00	1.00	0.52	0.48	0.72	0.04	0.16	0.08
	0.50	0.64	0.60	0.16	0.16	0.32	0.00	0.04	0.00
12	0.25	0.96	1.00	0.44	0.36	0.48	0.00	0.08	0.08
	0.50	0.44	0.56	0.08	0.08	0.16	0.00	0.00	0.00
14	0.25	1.00	1.00	0.28	0.20	0.32	0.00	0.08	0.08
	0.50	0.28	0.36	0.04	0.04	0.08	0.00	0.00	0.00
16	0.25	0.56	0.96	0.16	0.12	0.16	0.00	0.00	0.00
	0.50	0.16	0.24	0.00	0.00	0.04	0.00	0.00	0.00
18	0.25	0.48	0.92	0.08	0.08	0.00	0.00	0.00	0.00
	0.50	0.08	0.16	0.00	0.00	0.00	0.00	0.00	0.00
20	0.25	0.12	0.32	0.08	0.04	0.00	0.00	0.00	0.00
	0.50	0.04	0.08	0.00	0.00	0.00	0.00	0.00	0.00

Table 1: Performance of various ML models as n increases for ratio = 0.25, 0.5

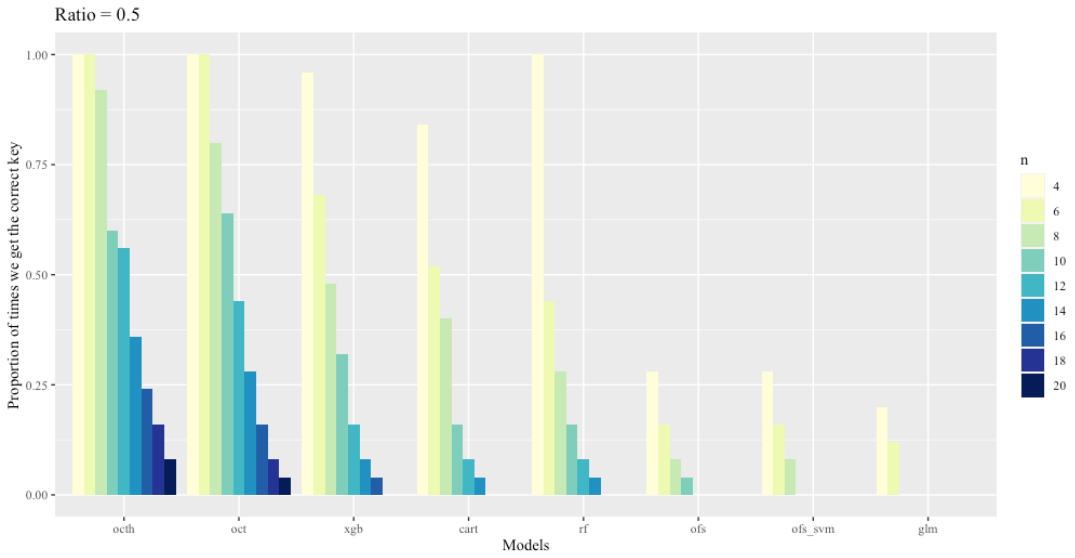


Figure 1: Performance Comparison of various ML models as n increases for ratio = 0.5

Figure 1 also shows a model-wise comparison as n increases at ratio = 0.5. To see the results for other ratios as well as the individual performance of each model, please refer to the Appendix Figure 4 and 5.

Observations:

- For ratio = 0, we always recover the key
- For the same n, it is difficult to recover the key with a higher number of 1s (higher "ratio") – Higher number of 1s implies more "features" to consider, and hence, it is more difficult to solve
- GLM, OFS, SVM: Perform extremely poorly – This is because of the high non-linearity (XOR Operation) in the data
- CART, RF, XGB: Perform moderately since they are able to capture some non-linearities of data
- OCT: Performs better than other traditional classifiers
- OCT-H: Outperforms all other models since it considers all the features where key bits are 1 at each hyperplane split as shown in Figure 2

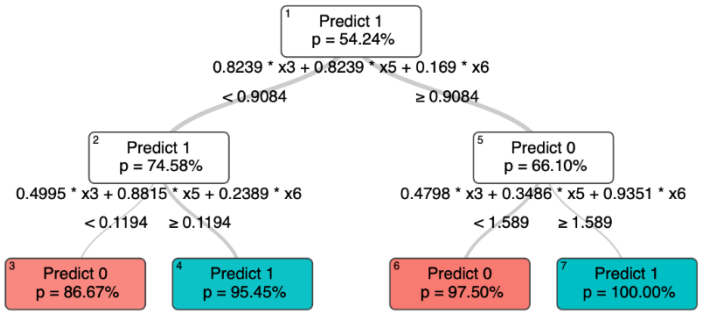


Figure 2: OCT-H for secret key s = 001011

For key s = 001011, OCT-H is able to detect the keys that are 1 and hence, the most important features since all the splits are made using information from Bit 3, Bit 5, and Bit 6 from matrix A.

MIO v/s Brute Force

Table 2 shows the comparison between MIO and Brute Force as n increases in terms of recovery rate and computational time at $ratio = 0.5$. We classify the first 15 $\{n, n + \frac{M}{50}, .. n + \frac{14M}{50}\}$ of our m's as Small, the last 15 $\{n + \frac{36M}{50}, \dots, n + M\}$ of our m's as Large and the rest $\{n + \frac{15M}{50}, \dots, n + \frac{35M}{50}\}$ of our m's as Medium and compute the average key recovery rate and computational times. We omit the results for $ratio = 0, 0.25, 0.75, 1$ in the interest of space but they are similar to these results.

n	m	Average Key Recovery Success Rate		Average Computational Time (seconds)	
		MIO	Brute Force	MIO	Brute Force
4	Small	0.93	0.00	0.0025	0.0000
	Medium	1.00	0.47	0.0039	0.0000
	Large	1.00	1.00	0.0054	0.0000
6	Small	0.87	0.00	0.0043	0.0000
	Medium	1.00	0.26	0.0116	0.0000
	Large	1.00	1.00	0.0167	0.0001
8	Small	0.93	0.00	0.0157	0.0000
	Medium	1.00	0.32	0.0316	0.0001
	Large	1.00	1.00	0.0530	0.0002
10	Small	0.87	0.00	0.0436	0.0000

	Medium	1.00	0.26	0.0844	0.0006
	Large	1.00	1.00	0.1311	0.0015
12	Small	0.93	0.00	0.1187	0.0000
	Medium	1.00	0.26	0.2744	0.0020
	Large	1.00	1.00	0.2785	0.0076
14	Small	0.87	0.00	1.5197	0.0000
	Medium	1.00	0.42	1.1087	0.0084
	Large	1.00	1.00	1.1418	0.0296
16	Small	0.87	0.00	1.9152	0.0000
	Medium	0.95	0.32	2.7076	0.0494
	Large	1.00	1.00	3.8381	0.1741
18	Small	0.93	0.00	19.6932	0.0000
	Medium	1.00	0.26	33.8801	0.2194
	Large	0.73	1.00	204.5913	0.8258
20	Small	0.80	0.00	8.0081	0.0000
	Medium	0.89	0.37	26.8057	1.3288
	Large	0.93	1.00	56.8756	4.4301

Table 2: Comparison of MIO v/s Brute Force at Ratio = 0.5 at different m for each n

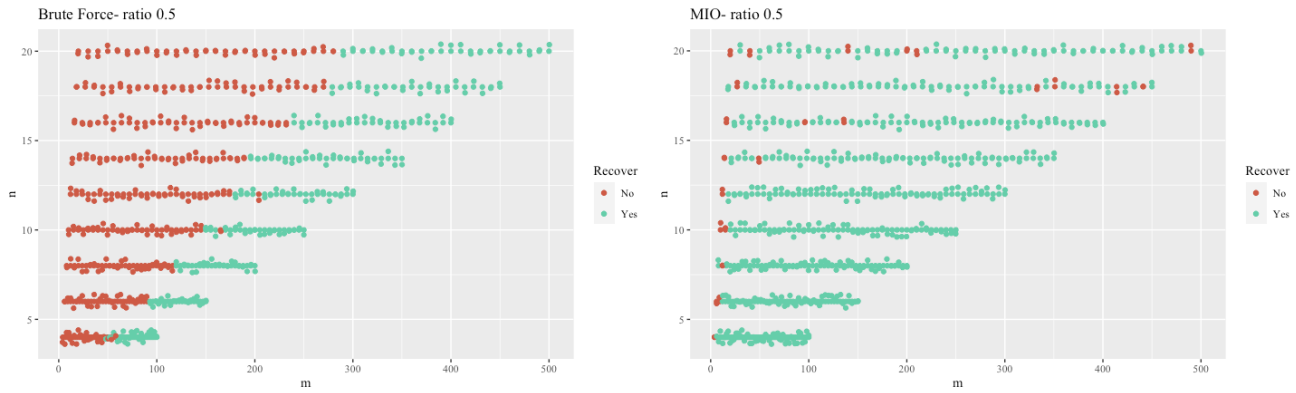


Figure 3: Comparison of Brute Force and MIO over different m for n = 4 to 20 at ratio = 0.5

Figure 3 also shows a model-wise comparison as n increases at ratio = 0.5. To see the results for other ratios, please refer to the Appendix Figure 6.

Observations:

- MIO is able to recover the key for much smaller number of examples (m) compared to Brute Force
- MIO is very slow compared to Brute Force

Note: All results are for Bernoulli error p=0.1.

CONCLUSION

We tried to solve the Learning Parity with Noise problem using ML and MIO. We conducted a series of experiments to analyze the key recovery rate and computational time over different key bit distributions of 0s and 1s, key lengths from 4 to 20, and a different number of samples depending on each key length. We found that most ML models are not able to recover the key as the ratio of 1s increases due to extremely high non-linearity in data except, OCT-H. This is likely because **OCT-H was able to identify the most important features (or bits)**

from our example matrix A and use those features in hyperplane splits. We also found that even though the MIO solution is quite slow, **MIO requires a much smaller number of samples to recover the secret key compared to Brute Force.**

CHALLENGES AND FUTURE WORK

1. **Recovering Keys for Higher Dimensions:** We were not able to run the ML models, MIO and Brute Force for higher n since either the training and cross-validation were taking very long to run or the algorithm run times were too high. We can try running these solutions on stronger processors to understand their performance for a higher number of bits.
2. **Recovering Keys for Higher 1:0 Ratio:** For ML models, we should try more complex models such as neural networks that capture highly non-linear relationships, especially for secret keys that have a higher ratio of 1s compared to 0s.
3. **Assessment Against Baselines/ Benchmarks:** There are several approaches that solve LPN given time, memory, and sample limitations using either Combinatorial or Decoding algorithms. Most of the basic approaches we came across use matrix inverse calculations on the Galois field which were not so straightforward to implement and hence, we decided to go ahead with a simple Brute Force baseline. However, we should try combinatorial and decoding algorithms and see how the MIO solution compares against them.

APPENDIX

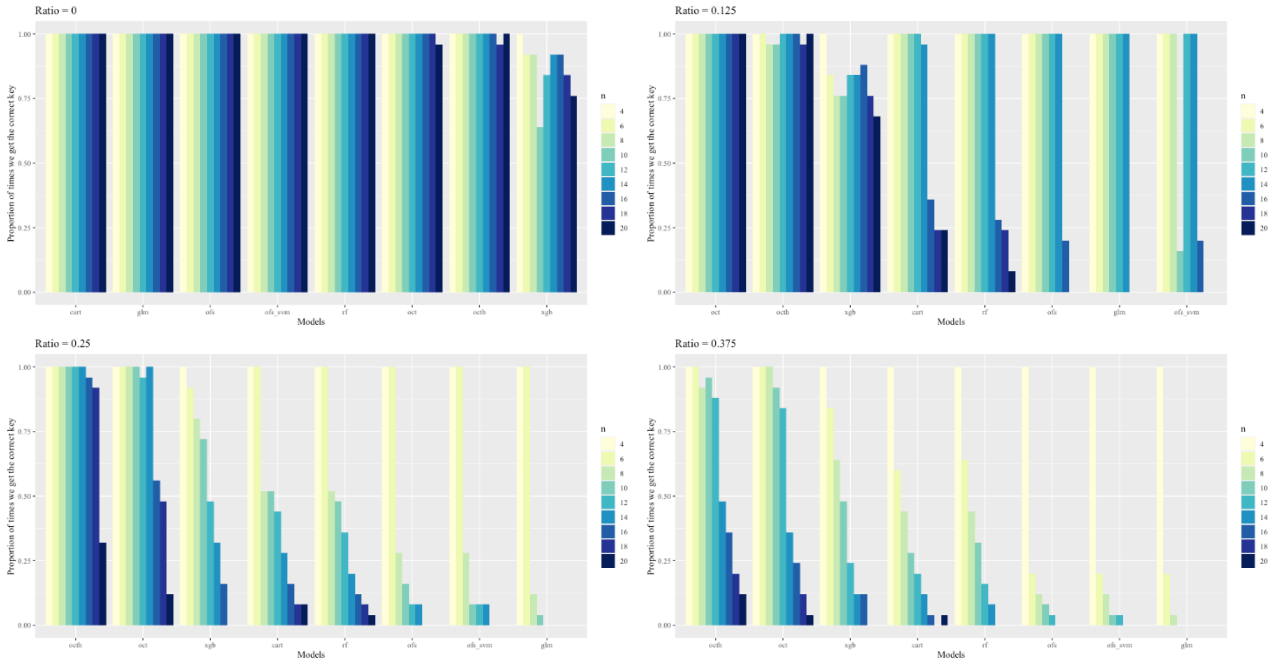
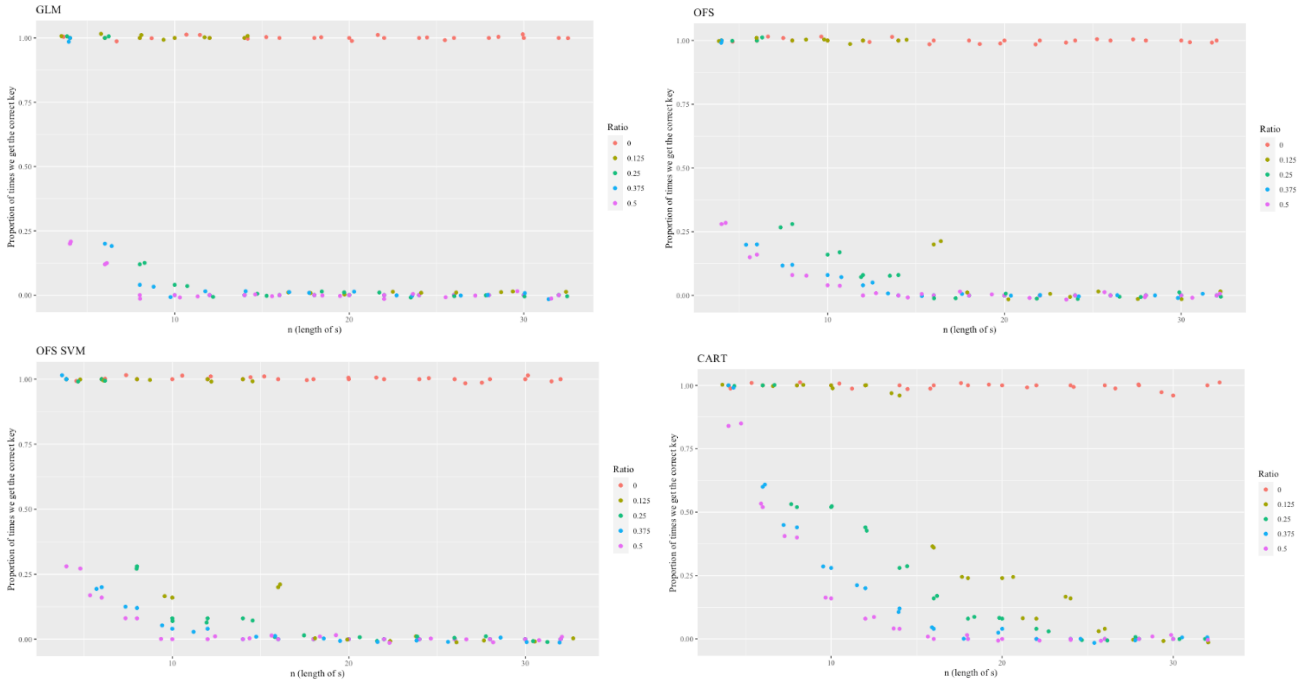


Figure 4: Performance Comparison of various ML models as n increases for ratio = 0, 0.125, 0.25, 0.375



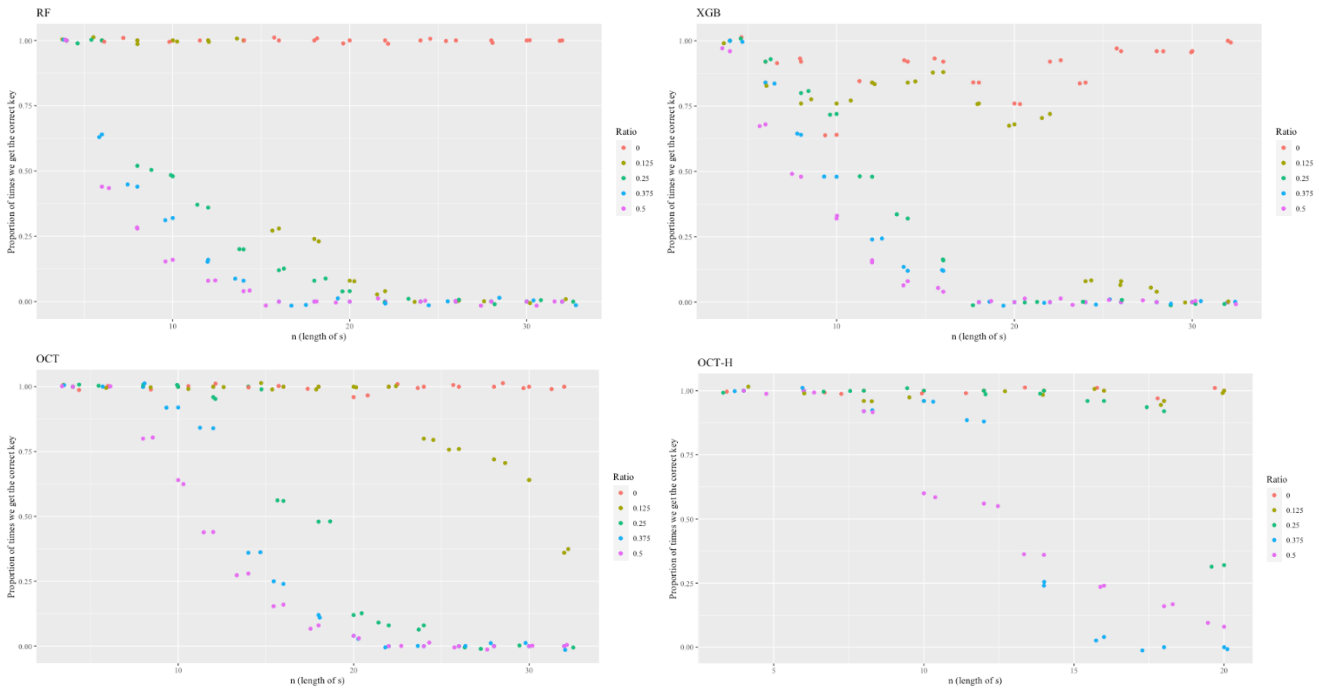
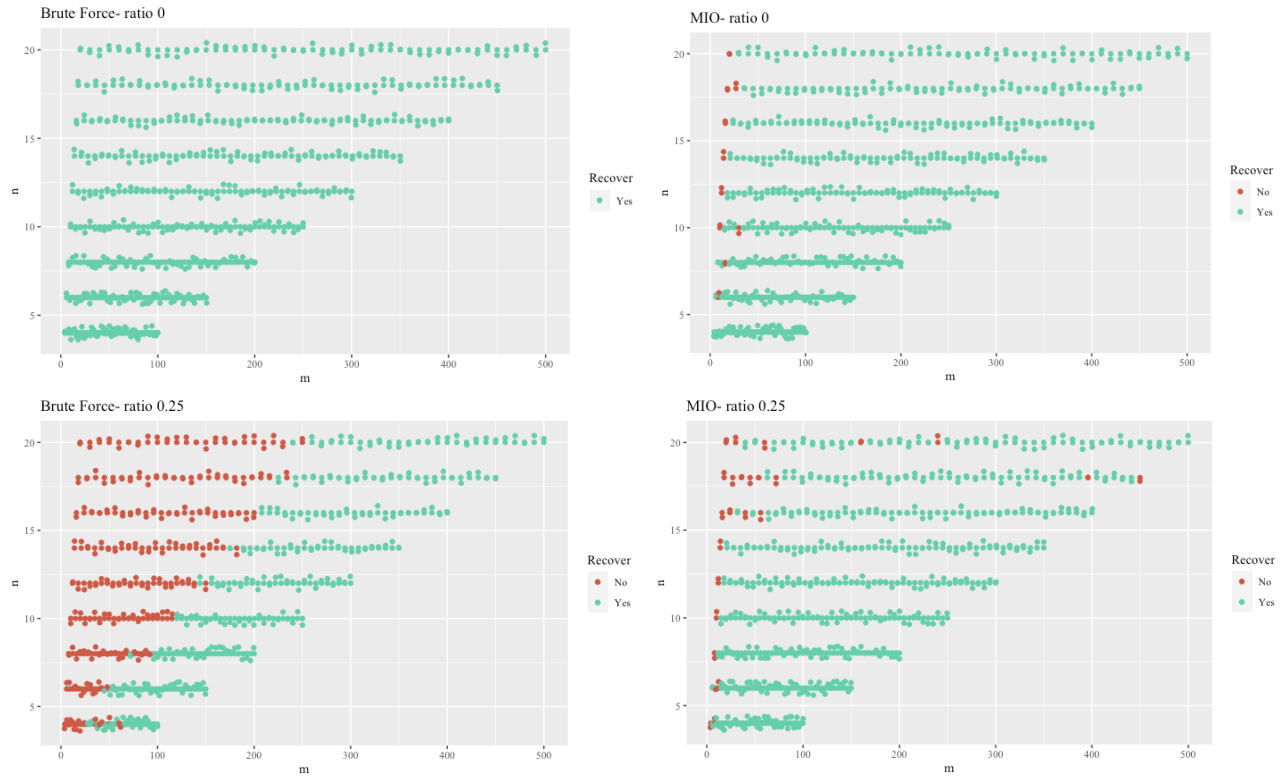


Figure 5: Performance Comparison of various ML models as n increases for ratio = 0, 0.125, 0.25, 0.375, 0.5



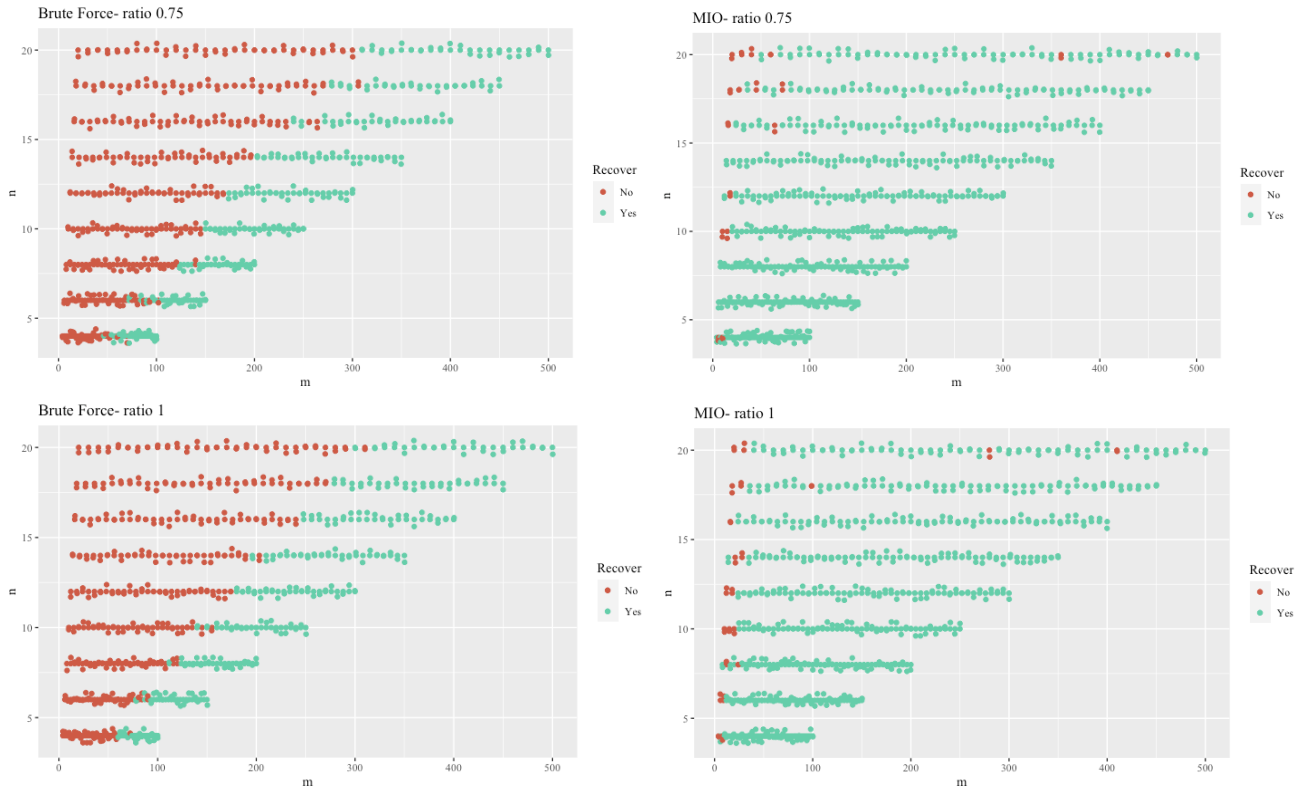


Figure 6: Performance Comparison between MIO and Brute Force at various n and corresponding m for ratios = 0, 0.25, 0.75, 1