# ECE236A: Linear Programming  Sparse - Designs for Linear Regression

Yaman Yucel UID: 605704529

**Introduction:**

As the course project, we were required to train a regressor model with the following loss functions. In Task 1, offline solution is represented. In Task 2, online solution is represented.

$$min \frac{1}{N} \|\gamma - (X\theta + b1)\|_1 + \alpha\|\theta\|_1$$

Since Equation 1 is linear, it can be represented as a linear program which is shown under Task 1.1

**Task 1.1:** Loss function can be rewritten as a LP in the following form.

$$min \left(\frac{1}{N}\right) 1^T t + (\alpha) 1^T z$$

$$s.t. \ -t \le \gamma - (X\theta + b1) \le t$$

$$-z \le \theta \le z$$

**Task 1.2:** In this task, LP in Task 1.1 is solved using cvxpy for multiple values of alpha. Firstly, using mean absolute error as the loss function yields to outlier selection since least squares estimates are highly sensitive to outliers. When data contains many outliers, one can get better results using the mean absolute error loss function. Same as the least squares solution, $l_1$ penalty yields to feature selection. When alpha gets very large, in order to minimize the Equation 1, weights should be zero. However, setting all weights to zero does not yield to a good regressor, since we also eliminate features that are useful for regression. Therefore, there is a tradeoff between the fit to the data and the $l_1$ norm of the weights.

**Results for Online News Popularity Dataset:** In Figure 1, optimal alpha that yields the lowest test error is found out to be 0.01. When alpha gets smaller, we do not do feature selection which yields train error and test error to be close to the solution without $l_1$ penalty. When alpha gets bigger, we start to eliminate features that are useful, which is problematic since elimination yields to poorly performing regressor. Test error with alpha = 0.01 is 0.2708 and train error is 0.2788.

**Results for Synthetic Dataset:** In Figure 2, optimal alpha that yields the lowest test error is found out to be 0.01. Test error with alpha = 0.01 is 0.61 and train error is 0.15.

**Task 1.3:** In this task, I have tried to decrease the number of features in offline case, by using the regressor variables found by training whole dataset. It can be easily observed from the Task 1-2, penalty term does the feature selection part. The weights that are bigger than other weights are more useful for the regressor, since our data is standard scaled. The weights that are nearly zero have no contribution to the regression. Therefore, by taking the k weights that are the biggest among the weights, one can easily select the percentage of features used in regression. k is chosen to be ceil(# of total features * percentage of features). I have also tried to use SelectKBest according to f_regression score and chi2 score, however since we do not do ordinary least squares regression. This method performed poorly. One can also try to backward and forward feature selection. However, since sparsity and $l_1$ penalty are the main goals of the project, I have chosen to use the $l_1$ penalty results. In backward feature selection, one discards one feature at a time and tries to get and decrease in the training error. In forward feature selection case, one starts with no features and adds one feature at a time that results in the lowest training error. These methods are computationally expensive, but they can be used for any machine learning method. These both methods also assume that each feature effect is independent of the other feature, which is a downside.

**Results for Online News Popularity Dataset:** In Figure 3, by selecting only 20% of features, we lower test error to 0.2729 and get very close result, test error 0.2707 when using all features. For this dataset, we can conclude that we can reach similar errors by using 20% of all features, but there is no increase in performance of the regressor when it is trained with less features.

**Results for Synthetic Dataset:** In Figure 4, by selecting only 20% of features, we significantly lower the train and test error from 16.74 to 0.3 and 17.07 to 0.48. When we use all features our test error is 0.61 which is higher when we use 20% of total features. Therefore, we can conclude that this dataset includes redundant and noisy features which disrupt the performance of the regressor.

**Task 1.4:** There are two purposes for using MAE loss. We can use linear programming solvers to solve the minimization of the loss function and MAE is used for regression analysis when there are many outliers present. To further improve the regressor, one can eliminate the outliers using the regressor variables obtained from the training with all data. The residuals will follow the normal distribution since our fit is assumed to be $y = \theta x + b + e$ , where e is the error term with N(0,1) probability distribution. Using large amount of data can decrease the impact of the error to the fitting, since error has a mean of zero. However, we do not require to use large amounts of when we discard the data that yields to high residuals. The training results would be same if data does not contain outliers. Therefore, I have selected k data that has the lowest residuals for fitting. k is chosen to be ceil(# of features * percentage of features). I have also tried random sampling, however test error was higher when we have less samples. One can use Cook's distance algorithm to determine samples, however we require to train total number of sample times to obtain the cook's distance. Cook's distance is also a method for ordinary least squares, which can be adapted to the MAE loss.

**Results for Online News Popularity Dataset:** By using only 1% of the sample, we can get test error of 0.2709, whereas training error is nearly zero, since we have discarded the outliers. As we add the outliers to the training samples, the test error does not change and stays at 0.27 whereas the training error approaches to the value found in the Task1-2, since outliers plays role in the training error. These data can be easily observed from the Figure 5.

**Results for Synthetic Dataset:** In this case, we observed that by using only 50% of the sample, we can get test error of 0.74, whereas training error is nearly zero, since we have discarded the outliers. As we add the outliers to the training error, the test error does not change and stays at 0.70 whereas the training error approaches to the value found in the Task1-2, since outliers plays role in the training error. These data can be easily observed from the Figure 6. It is noteworthy to mention that 50% sample is equal to the number of features in this case. We get a good regressor when we have samples as much as the number of features. When we have less samples, the X matrix is underdetermined and yields to multiple solutions which effects the performance of the regressor. When we have equal number of samples and features, we nearly have a full row rank matrix since the X matrix is sparse and normally distributed. Solution yields to a unique solution which yields a good regressor.

**Task 1.5:** For this problem, I have tried to combine methods for Task 1.3 and Task 1.4.I wanted to do equally amount of feature selection and sample selection, so the percentage of samples and percentage of features are chosen to be $\sqrt{r}$ where r is communication cost ratio. By doing experiments, I have noticed that doing a feature selection after sample selection make your regressor to be trained with less communication than doing sample selection after feature selection. The difference can be seen from the Figure 7 and 8. Note that gaussian data is used to obtain these plots. Rather than choosing $\sqrt{r}$ one can weight one method more than the other method, which may lead to better communication for some dataset, but less communication for other datasets.

**Results for Synthetic Dataset:** From the Figure 8, we can observe that using 10% communication cost, we can train a similar regressor to the 100% communication cost. This implies the power of feature selection and sample selection algorithms. Although we require much data to train good regressor, one can use less communication cost and obtain good regressors. This dataset contains irredundant features since it has only 600 samples but contains 500 features. Most of the features are eliminated by the algorithm which reduced the communication cost significantly.

**Results for Online News Popularity:** From the figure 9, it can be easily seen that by only using the 1% of the communication cost, one can train a same regressor to the 100% communication cost. This dataset contains irredundant data since it has only 58 features but contains 3964 samples.

**Task 2:** For this problem, one might hold each data at the central node and retrain the model from the scratch using the Task 1-2 algorithm. However, this method is computationally expensive especially when the sample size inside the central node gets bigger and bigger as times goes on. Therefore, I proposed a new method which is suitable for online learning purposes. Stochastic Gradient Descent is an optimization tool when we do or do not access to whole data, or we want to train our regressor in multiple batches. SGDRegressor from the sklearn library provides the optimization tool with the required loss function and penalty term, when the loss is selected as epsilon_insensitive and epsilon = 0, penalty = 'l1', alpha = alpha found in previous parts. Different than the offline algorithm, online algorithm requires a learning rate which specifies the speed of the convergence. The learning rate can be chosen as adaptive where learning rate gets closer when error from the loss function drops. Although sklearn implementation might differ from the basic equation written below, the equation is the core of the mentioned algorithm.

$$\theta_n = \theta_{n-1} + lr * sgn\big(y - (x^T\theta + b)\big)x + \alpha\, sgn(\theta)$$
$$b_n = b_{n-1} + lr * sgn\big(y - (x^T\theta + b)\big)$$

I also do online version of sample selection and feature selection while satisfying the communication constraint at each time step. At first, in order to get a glimpse of the regressor, I do not do any feature selection or sample selection. After getting a reasonable regressor, algorithm starts both selection methods to be able to process more data under communication constraint. For sample selection, I use z-score method on the residuals of the new data. If the absolute z-score of the residual of the new data is less than 3 which is also a tunable parameter, I consider data as the outlier and do not send the data to the central node. For feature selection, method if the corresponding weight for feature i is nearly zero (10**-5, tunable parameter), I do not use that feature to train my regressor. When I do not send new sample or feature, I gain space to send more sample which results in a better trained regressor. However, this algorithm involves many tunable parameters. Note that, choosing where to start selection methods is also tunable parameter.

**Results for Online News Popularity:** The advantage of using SGD is that the training process is computationally inexpensive, therefore method is suitable for large amounts of data. However, when the amount of data is low, SGD performs worse than the offline method due to the convergence issue. Using communication constraint of 0.2, I was able to get test error to 0.41 which is slightly higher than the offline method. These results are available in the Figure 10.

**Results for Synthetic Dataset:** For this method, I require more data to train a better regressor since I was not able to converge the SGD algorithm with the given amount of data. Therefore, using a communication constraint as 100% yield to a test error of 12.17. These results are available in the Figure 11. Note that, better hyperparameters can yield to a lower test error.

**Notes about code and experiments:** Each task can be resimulated with Sample_Experiment_yamanyucel.ipynb. I have added multiple attributes to the object and changed input of the methods. Self.selectedColumns is created for to be able to access the features and weighs selected outside the class. self.learningRate is added for the online learning process which should be tuned with alpha and an hyperparameter search algorithm for each each dataset, otherwise online learning algorithm might not converge. self.communicationCostRatio is added for debugging purposes which shows whether constraint is provided is followed or not. For Online news Popularity, learning rate is found out to be 0.001 and for synthetic dataset, learning rate is found out to be 0.025. If the sample code is used, there is no need to adjust learning rate in the code. Note that both alphas are taken to be 0.01. For task, 1.3 and 1.5, one might need to adjust the columns of test data according to the self.selectedColumns otherwise, sizes of the matrix multiplication will give an error.

**Appendix:**

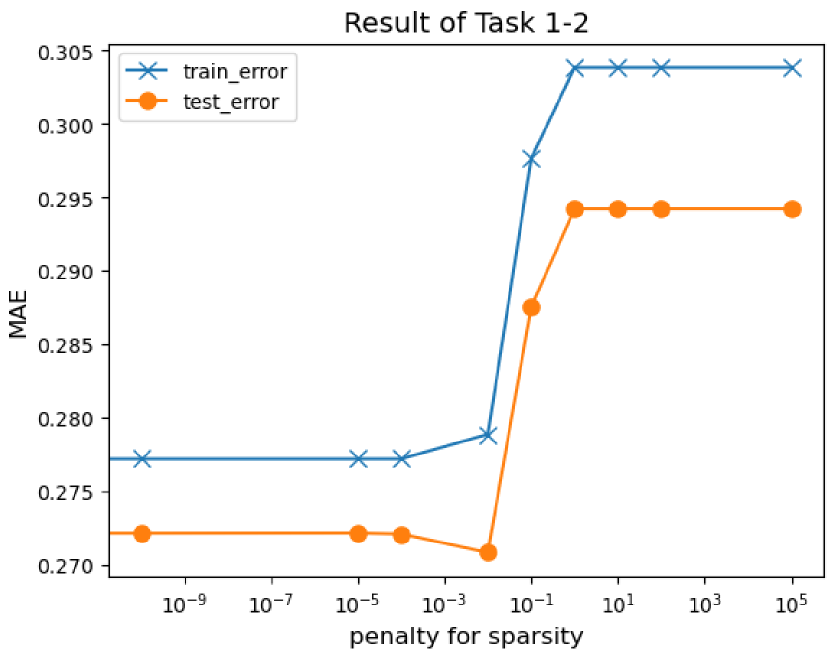

Figure 1: Result of Task 1-2 on Online News Popularity dataset

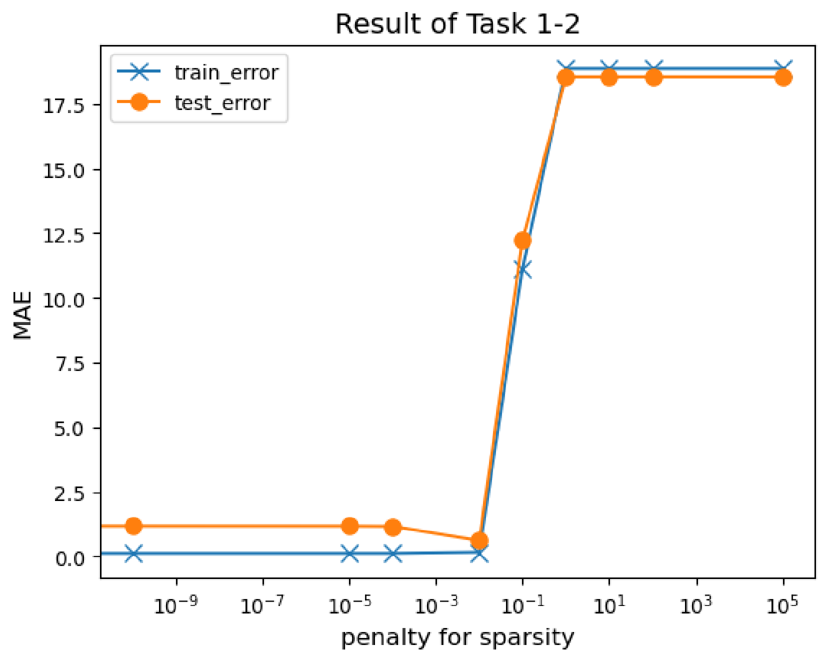

Figure 2: Result of Task 1-2 on Synthetic dataset

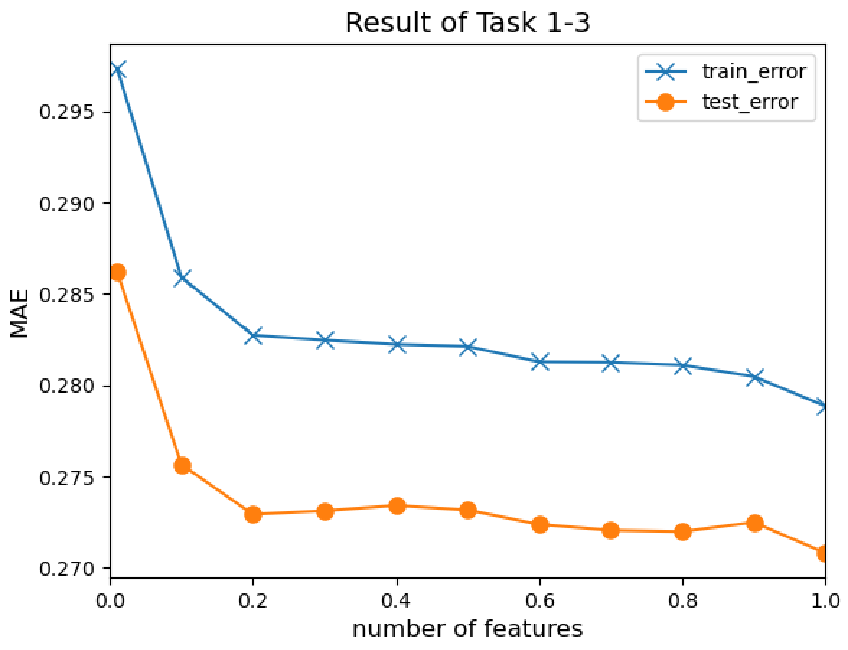

Figure 3: Task 1-3 on Online News Popularity Dataset

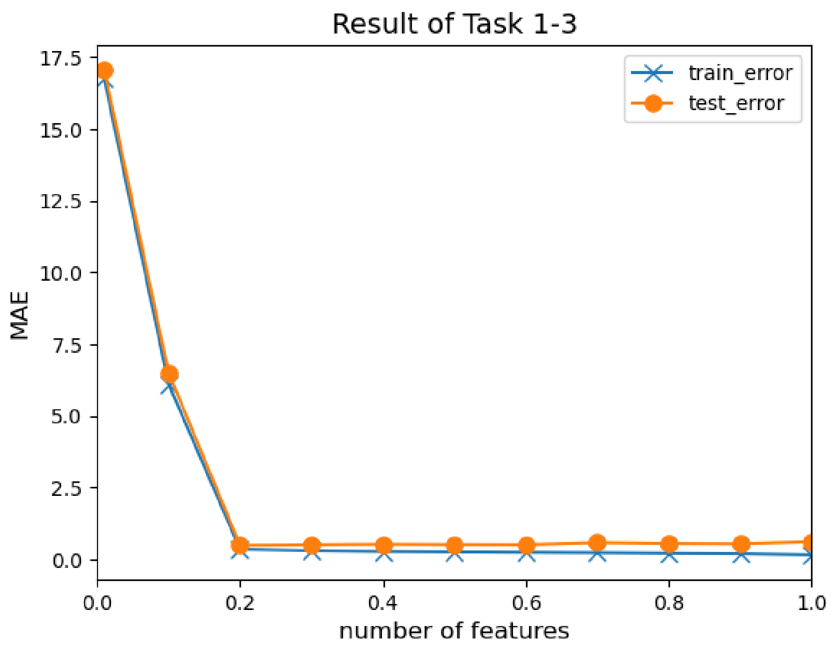

Figure 4: Task 1-3 on Synthetic Dataset

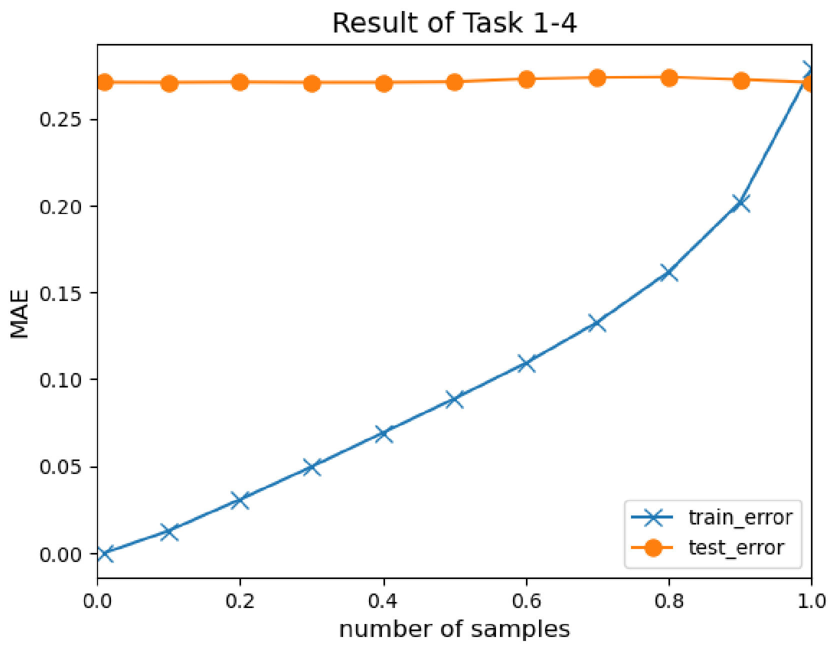

Figure 5: Task 1-4 on Online News Popularity Dataset

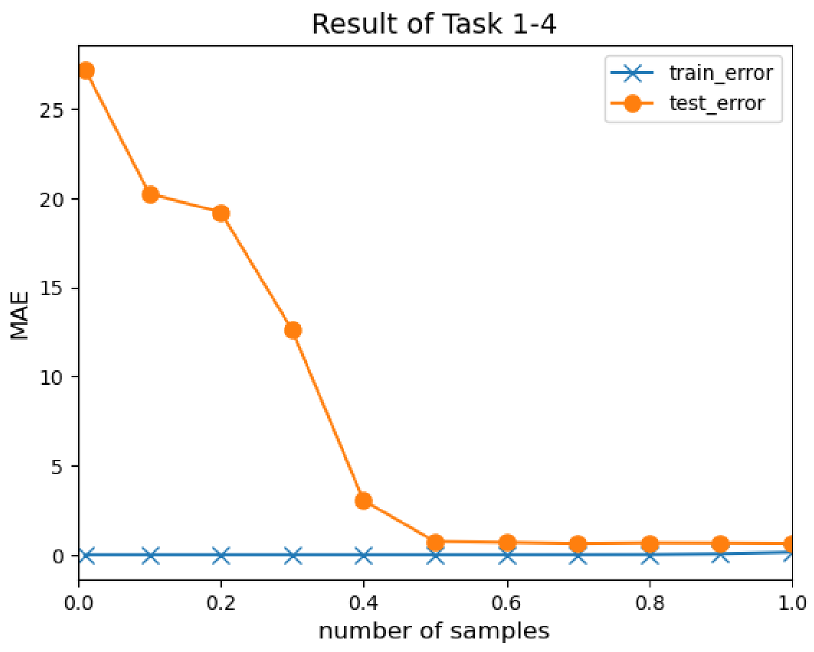

Figure 6: Task 1-4 on Synthetic Dataset

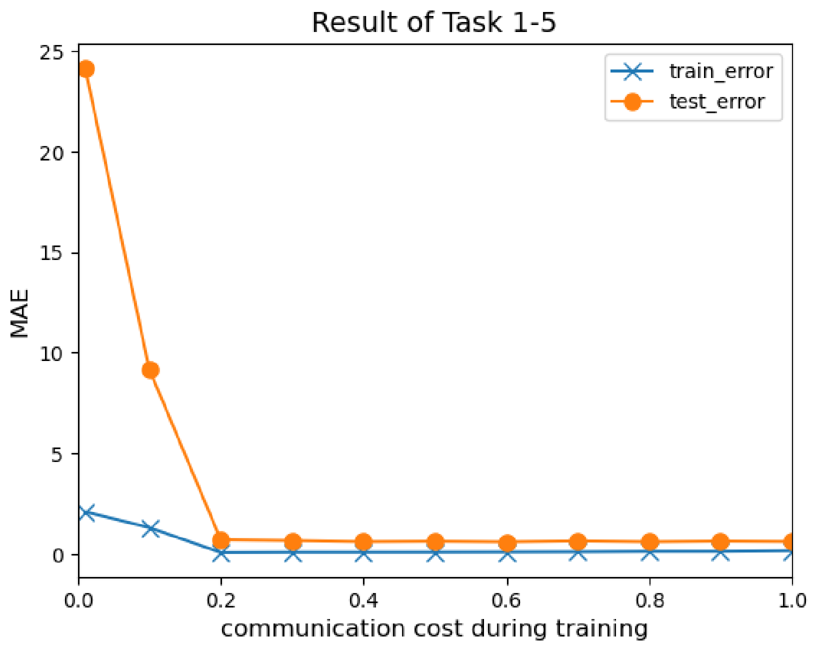

Figure 7: This is an extra plot, first sample selection then feature selection for Synthetic Dataset.

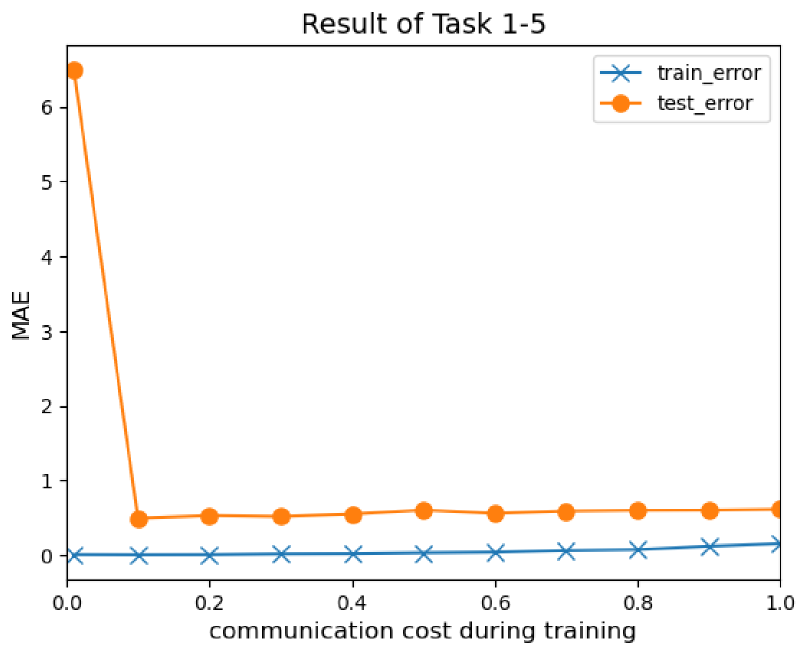

Figure 8: Task 1-5 on Synthetic Dataset

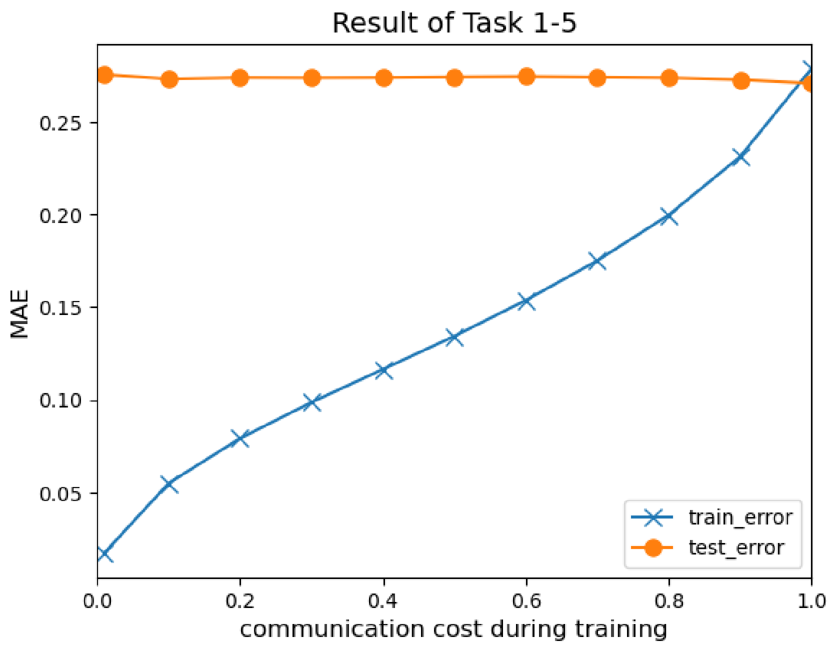

Figure 9: Task 1-5 on Online News Popularity Dataset

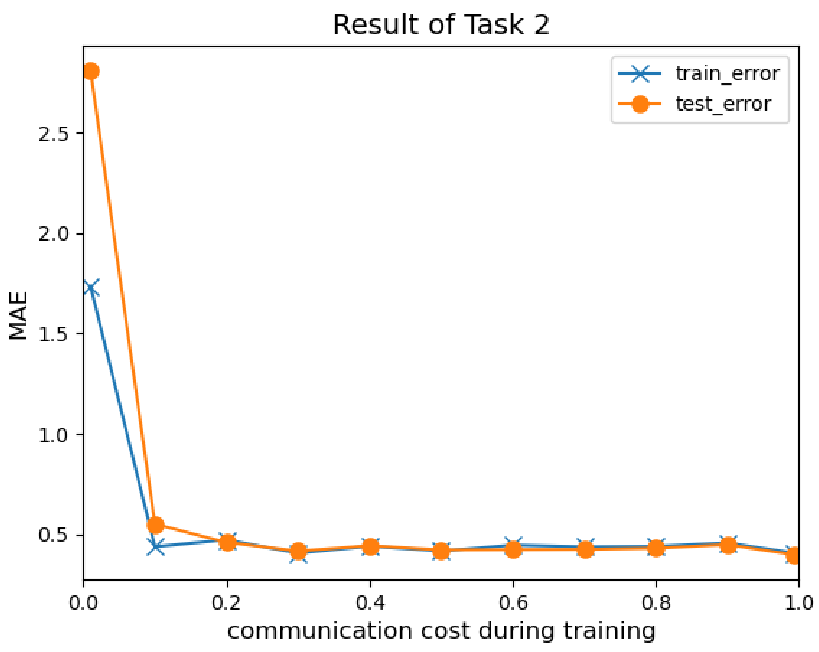

Figure 10: Task 2 on Online News Popularity Dataset

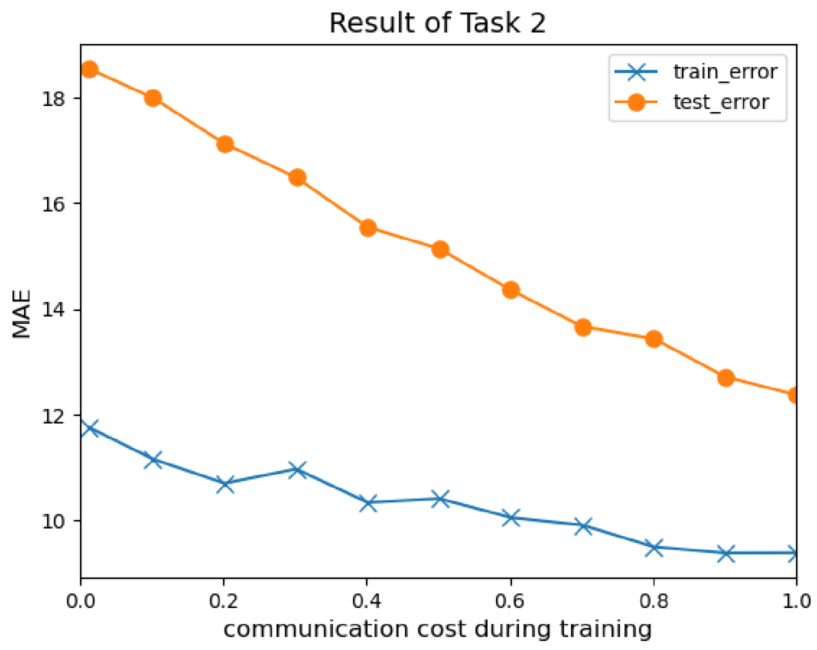

Figure 11: Task 2 on Synthetic Dataset