Fall 2022
ECE 236A
Professor: Christina Fragouli
TAs: Merve Karakas and Xinlin Li

ECE236A: Linear Programming

Sparse Designs for Linear Regression

Due: Sunday, Nov 6$^{\text{th}}$, 2022, before 11am.

To be completed by teams of 1-2 people.

Project Description:
In this project, you will explore the "usefulness" of data samples and features for training a sparse linear regressor. This is an open-ended project, which means there may be some questions with no "right" answers, and that we appreciate innovative design ideas. Your main constraint is that you need to use Linear Programming formulations.
You can work in teams of one (preferred) or two people. The project is due on **Sunday, Nov 6th before 11 am** and needs to be uploaded on Gradescope.

1 Background

Linear Regression attempts to model the relationship between a vector and its label using a linear predictor function. More specifically, let $x \in \mathbb{R}^M$ be a vector of M entries (we will refer to the entries as *features*), and let $y \in \mathbb{R}$ be its label. A regression task is comprised of computing $\hat{y} = g(x)$ which is an estimate of y. $g(x)$ is called a linear regressor when a linear transformation function is used to extract information from the input vector x, i.e., $g(x) = \theta^T x + b$. The unknown parameters $\theta \in \mathbf{R}^M$ and $b \in \mathbf{R}$ are called weights and bias, respectively. The learning task in linear regression is to estimate the parameters θ and b leveraging a given set of potentially noisy data, i.e., from given (x^i, y^i) pairs, $i = 1, \dots N$, that we assume satisfy $y^i = \theta^T x^i + b + n^i$, with n^i a noise variable. In Fig. 1, you can find an example of a dataset with one feature and corresponding labels. The red line is the learned line of the linear regressor when trained with this specific dataset.
Examples:

- Predicting the weight based on a person's height and gender. Since each person (a data sample) has 2 features (height and gender), x will have 2 entries (features).
- Predicting the temperature of the next day using the temperature of the last 120 days and the temperature of that day in the previous 3 years. Each x will have 123 entries (features).

Assessing Regressor Performance. A good regressor gets close to the actual label for most input vectors. One way to measure the quality of a regressor is through loss functions. One of the

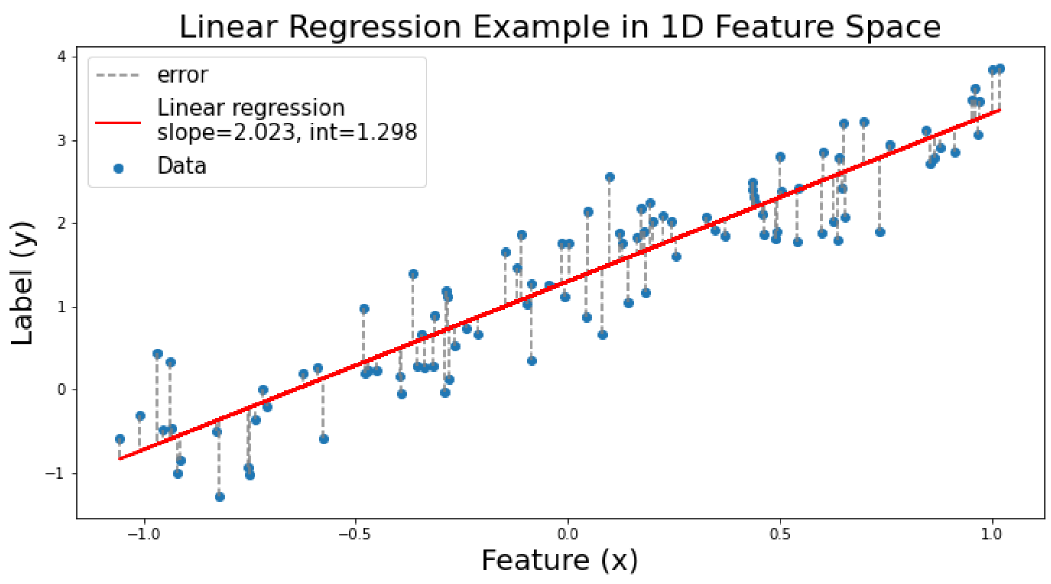

Figure 1: One-dimensional example for linear regression. $y^i = 2x^i + 1.3 + n^i$ where n^i is noise generated from $\mathcal{N}(0, 0.5)$

most commonly used loss functions is the Mean Absolute Error (MAE). Specifically, let $\mathcal{X} \subseteq \mathbb{R}^{N \times M}$ be a matrix that collects N data samples x^i and $\mathcal{Y} \subseteq \mathbb{R}^N$ be a vector collecting the corresponding labels y^i. Then the MAE loss for linear regression is defined as

$$\mathcal{L}(\mathcal{X}, \theta) = \frac{1}{N} \sum_{i=1}^{N} |y^i - \hat{y}^i| = \frac{1}{N} ||\mathcal{Y} - (\mathcal{X}\theta + b\mathbb{1})||_1 \tag{1}$$

Sparsity: When the number of features is large, it may be possible to achieve accurate prediction using a small subset of informative features. The problem of finding a θ with a small number of nonzero entries is called sparse linear regression. In this problem, on top of finding parameter values that offer a good fit for the data, we also want to find a sparse weight vector. For instance, considering the second example above, it is possible that we can get a similar accuracy using the temperature of the last 10 days and the temperature of that day in the previous 3 years. The challenge is that we do not know this in advance: it may be that we need to use the temperature of the last 13 days, or it may be that it is more useful to "skip" some past days, e.g., keep the temperature of past days 1, 2, 3, 5, 8, etc. We thus want to formulate an optimization problem that helps us identify such sparse structures if they are possible.
To promote sparsity while minimizing the loss, we can introduce an l_1-norm as a penalty and use the following problem to find a sparse linear regressor:

$$\arg\min_{\theta, b} \frac{1}{N} ||\mathcal{Y} - (\mathcal{X}\theta + b\mathbb{1})||_1 + \alpha ||\theta||_1 \tag{2}$$

where $\alpha \geqslant 0$ is a regularization parameter that controls the trade-off between the fit to the data and the l_1-norm of the weights. If α is very large, all coefficients are equal to zero. If α is very small, then the problem becomes equivalent to the MAE estimate. For values in between, it can yield models with different sparsity and regressor performances.

Training a Linear Regressor. In supervised learning, we assume that a relatively large set of inputs (called *training data*) $\mathcal{X}_{\text{Train}}$ and their corresponding labels $\mathcal{Y}_{\text{Train}}$ are available to us. The regressor is then trained in some way to determine the best parameters such that these training data are correctly labeled, i.e., the parameter θ and b is chosen so that the quantity $\mathcal{L}(\mathcal{X}_{\text{Train}}, \theta)$ is sufficiently low. The ultimate goal of the regressor is clearly not to correctly label $\mathcal{X}_{\text{Train}}$, since for these points the correct labels are already known. However, the premise is that if the sets $(\mathcal{X}_{\text{Train}}, \mathcal{Y}_{\text{Train}})$ are good representatives of the type of input vectors that are going to be labeled, the regressor would still perform closely to $\mathcal{L}(\mathcal{X}_{\text{Train}}, \theta)$ for these input vectors as well. The performance of the regressor is then evaluated on a data not encountered before (as part of the training test), which we denote as $(\mathcal{X}_{\text{Test}}, \mathcal{Y}_{\text{Test}})$. In the data we provide, we are going to specify which will be kept aside for testing and which will be used for training.

2 Project Goal and Details

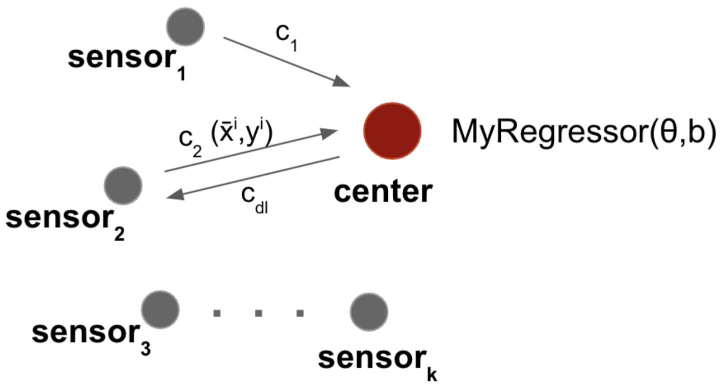

Figure 2: Example of a distributed sensor setting with k sensor nodes.

In this project, we consider a central node that wishes to train a sparse linear regressor by collecting (labeled) data x^i from distributed sensors; that is, a sensor collects the feature vector x^i and its associated label y^i and sends $(\bar{x}^i, y^i)$ to the central node. We assume that $\bar{x}^i$ is some function of x^i, for example, it can be a subset of the features. The sensors communicate with the central node over channels that are communication constrained: there is a cost associated with each data sample transmission. We define the total communication cost during training as $N_{train} * M$ where M is the number of features sent for each data sample and N_{train} is the number of sent data samples.

We divide the project into two parts: in Task 1, we assume the central node has access to the whole dataset, while in Task 2 we consider the distributed setting of Figure 2 with communication constraints. We also provide two datasets for evaluation; however, the algorithms you find for each task should be applicable to any dataset that can be used for linear regression. You should perform the given tasks for both datasets and provide plots for each of them in your report as described next.

Task 1: Offline Solution (7 points)

In this part, we assume that the central node has already collected and has the full knowledge of the dataset. The goal is to find a sparse vector θ that offers good regression performance. We can try to solve this problem by optimizing Eq. 2. Tasks 1-3, 1-4, and 1-5 are steps towards the distributed case (Task 2) where you will need to decrease the communication cost.

- **1-1 Reformulation (1 point)**
  Reformulate Eq. 2 as a linear program. Include the reformulated equation into your project report.

- **1-2 Train a Regressor (2 points)** Implement an algorithm to train a regressor by solving the linear program you formulated in Task 1-1. Explore the effect of l_1-norm penalty by changing the value of α and re-training the regressor. Select at least 5 different α values from a reasonably large range based on your experiments. Plot α vs training and test error and discuss your observations in the report. Recall that if we have N_1 data for training and N_2 data for testing, the training error is calculated as $\frac{1}{N_1}\sum_{i=1}^{N_1}|y^i - \hat{y}^i|$ and the testing error similarly.

- **1-3 Feature Selection (1 point)** By analyzing your results from Task 1-2 and the logic behind Eq. 2, propose your own method to reduce the number of features used for training the regressor. Your method should be adjustable to reduce the number of features to any specified level (e.g. 10%, 20%, etc.). Implement your method and use the selected features to train a new regressor. Plot the percentage of features used vs training and test errors. The choice of the percentage of features should cover at least 5 levels: around [1%, 10%, 30%, 50%, 100%].

- **1-4 Sample Selection (1 point)** Some data samples may not be useful for training the regressor, i.e., you may get the same regressor or one that gives a very similar error even if you exclude those data samples. Try to come up with a heuristic or a linear program that decreases the number of data samples used for training based on the training results you got in the Task 1-2. Plot percentage of samples vs training and test errors. The choice of the percentage of samples should cover at least 5 levels: around [1%, 10%, 30%, 50%, 100%].

- **1-5 Sample and Feature Selection (2 points)** You will here combine both the previous methods to further decrease the amount of information used for training the model. You can either do sample selection after fixing the number of features based on Tasks 1-3 and 1-4 or you can jointly optimize the number of features and samples used. You can consider other ways to enforce sparsity instead of l_1-norm. Implement one and tune your methods to satisfy different communication constraints. Plot communication cost during training vs training and test error. The choice of the percentage of samples should cover at least 5 levels: around [1%, 10%, 30%, 50%, 100%].

Note: *If you are a group of two, you are required to come up with, implement, and compare TWO different algorithms for questions 1-3, 1-4, and 1-5.*

Task 2: Online Solution (3 points)
The challenge in this part is that each sensor receives and sends its data in an "online" manner. Therefore, the decision to send a sample x^i or which features of the sample to send is performed without the knowledge of all the points in the training dataset. For simplicity we assume we have a single sensor node connected to the central node through a communication limited channel. We assume the central node can broadcast any information to the sensor node without any communication constraints; for example, it can communicate its current estimates of the model parameters. That is, we have uplink communication constraints (from sensor to central node) but no downlink communication constraints (from central node to sensor). More specifically, time is slotted, and at each time slot i, the sensor observes one data sample x^i as well as potentially some communication from the central node. The sensor has sufficient memory to keep all past data samples (however, cannot perform linear regression itself). Thus the sensor sends to the central node an $\bar{x}^i$ that can be a function of x^i, past data samples x^j, $j = 1 \dots i-1$, and the information communicated from the central node. Our goal is to have the central node train a good linear regressor while minimizing the communication cost.
Either utilize your solution in Task 1 if possible and reformulate it to use for online training or propose a new algorithm for online training. Plot communication cost during training vs training and test error. The communication cost here is measured in percentage and the choice in the plot should cover at least 5 levels: around [1%, 10%, 30%, 50%, 100%].

Note: *If you are a group of two, you are required to come up with, implement, and analyze TWO different algorithms for this task.*

3 Data set

You will need to validate your algorithms and implementations on two datasets:

1. Synthetic Dataset: We generated $N = 1000$ samples with dimension $M = 500$ and the sparsity of weight vector is 80%. Each feature is generated from an i.i.d. standard normal. 600 samples are used for training and the remaining 400 are used for testing. Labels are generated using $y = \theta^T x + b + n$ where θ is the sparse weight vector, x data, b bias, n noise, and y is the label vector.

2. Online News Popularity Dataset[1]: The dataset we will use contains 6608 data samples. Each sample corresponds to 58 numerical features of a news. The goal is to predict the number of shares in social networks, which stands for popularity. 60% of the data will be used for training and 40% for testing.

[1]https://archive.ics.uci.edu/ml/datasets/online+news+popularity

Note: This part is to inform you about which datasets you will experiment on. You do not need to code anything for getting or splitting datasets, you will just call the corresponding functions to get the datasets.

4 Implementation Requirements

Please download the project_code.zip file from Bruin Learn, which contains three files:

- We provide some utility functions in **utils.py**. You do not need to do any data cleaning, preprocessing, or splitting. You can get the processed datasets using functions `prepare_data_gaussian()` and `prepare_data_news()`. You can get the requested plots for each task using `plot_result()` function, for which you need to store the required experiment output into dictionaries and pass it to the function. You can find the input/dictionary format in the comment lines within the function. Do NOT change the code in this file.
- The online news popularity data is stored in **OnlineNewsPopularity.zip**. Unzip it before using `utils.prepar_data_news()` function.
- **MyRegressor.py** contains the skeleton implementation of a linear regressor. You are supposed to complete the missing parts according to the task instructions and submit it as **MyRegressor_{groupnumber}.py**. It is implemented as a class named `MyRegressor` with the following attributes and methods:
  - `self.weight` stores the value of θ in Eq. 2
  - `self.bias` stores the value of b in Eq. 2
  - `self.alpha` stores the value of α in Eq. 2, which is assigned at the time of initialization.
  - `self.training_cost` stored the communication cost ($N_{train} * M$) during training.
  - `train(self, trainX, trainY)`: this is where you will implement Task 1-2 that trains your Regressor. It takes the following inputs:
    - `self`: this is a reference to the MyRegressor object that is invoking the method.
    - `trainX`: this is a matrix of dimensions $N_{train} \times M$.
    - `trainY`: this is a vector in length N_{train}.
    - This function should output the mean absolute error on the training set and update the values of the class attributes.
  - `select_features(self)`: this is where you will implement Task 1-3 that selects informative features. It takes the following inputs:
    - `self`: this is a reference to the MyRegressor object that is invoking the method.
    - This function should output a list containing the indices of the selected features, e.g. if the first and 100th features are selected, it should output $[0, 99]$, and update the values of class attributes if applicable.

– `select_sample(self, trainX, trainY)`: this is where you will implement Task 1-4 that selects informative data samples. It takes the following inputs:
  - `self`: this is a reference to the MyRegressor object that is invoking the method.
  - `trainX`: this is a matrix of dimensions $N_{train} \times M$.
  - `trainY`: this is a vector in length N_{train}.
  - This function should output a subset of training data and corresponding label vector, and update the value of the class attributes if applicable.

– `select_data(self, trainX, trainY)`: this is where you will implement Task 1-5 that jointly optimizes the training cost. It takes the following inputs:
  - `self`: this is a reference to the MyRegressor object that is invoking the method.
  - `trainX`: this is a matrix of dimensions $N_{train} \times M$.
  - `trainY`: this is a vector in length N_{train}.
  - This function should output a subset of training data and corresponding label vector, and update the value of the class attributes if applicable.

– `train_online(self, trainX, trainY)`: this is where you will implement Task 2 which trains your regressor in an online manner. It takes the following inputs:
  - `self`: this is a reference to the MyRegressor object that is invoking the method.
  - `trainX`: this is a matrix of dimensions $N_{train} \times M$.
  - `trainY`: this is a vector in length N_{train}.
  - This function should output the total training cost and the mean absolute error on the training set. It also updates the values of the class attributes.

Notes:

1. You are allowed to implement other auxiliary methods/functions, but each task should be completed within the specified methods. You may also consider adjusting the input of methods. When you do so, please make sure it is generic and specify the meaning in the comments, so that we are able to reproduce the results or run verification on other datasets using your code.

2. When calculating the training/test errors, use mean absolute error (MAE) Eq.1, and do NOT include any penalty term (i.e. l_1-norm)

3. Do NOT use test data ($\mathcal{X}^{\text{Test}}, \mathcal{Y}^{\text{Test}}$) during training.

5 Report

Beyond the code, we also expect a report of up to 2 pages (up to 3 pages if you are a group of two) (excluding appendix), that describes in a complete way what is the rationale you used to decide which data samples or which features were sent as well as the specific algorithm description. Put the required figures in appendix to show your experiment results on two given datasets (10 figures in total for a group of one, 18 figures in total for a group of two), and discuss your observations. You are welcome to add extra plots if they help with your illustration. We also would like you to compare the performance of the regressor you trained, against the performance of the regressor that uses the full training dataset, and report the savings in terms of the trade-off between training cost and the training/test error.

6 Grading

- 10 points: This project will be graded mainly based on correctness and completeness. For Task 1-1, we will grade based on the correctness of the reformulation. For the remaining tasks, we will grade based on how you justified the algorithms you came up with.
- 5 bonus points: Bonus points will be given based on the creativity and performance of your proposed algorithms. 5-10 groups will be rewarded 5 extra points, subject to also presenting their approach on Nov. 17th.
- You need to submit a folder named `'group_{your_name}'` (e.g., `group_merve` for a group of one, `group_merve_xinlin` for a group of two) on Gradescope. Inside this folder there should be a file named `MyRegressor_{your_name}.py` (`MyRegressor_merve.py` for a group of one) as well as your report. It is optional to put the code of your experiments.