Fall 2021
EE 236A
Christina Fragouli
Mine Dogan and Kaan Ozkara

EE236A: Linear Programming

Project Part II - Bounds on Data Compression
Due: together with part I on Dec. 6

Project Description:
In the first part of the project, you explored the "usefulness" of data points for training a linear classifier at a central node, when the data points are collected in an online manner from distributed sensors. You developed and evaluated an algorithm, and you were (hopefully) able to significantly reduce the communication cost. A main challenge was that, each sensor needed to decide in an "online" manner, whether to send a specific data point it had observed or not, without knowledge of the remaining points in the training dataset.

We now wish to understand how good your algorithm is. To do so, we observe that clearly, if a genie were to give the whole data sets to the central node, and then the central node was able to decide which points to keep and which to throw away, we could do at least as well and potentially much better than using the online algorithm to decide which points to keep for classification. (Indeed, you could always run the same algorithm as before, assuming that the points arrive at some abritrary order, but potentially you can do much better than that).
Accordingly, we are asking you to develop an Integer Linear Program (ILP), that takes as input all the data points from two classes of data. The solution will tell you which data points to keep and which not, to train your classifier. We are also asking you to solve this Integer Linear Program through LP relaxation and then construct an integral out of the real solution.
For the ILP, you need to decide the objective and constrain functions. It is up to you to decide these, keeping in mind that your goal is to minimize the number of points used while maintaining the classification accuracy.
We next give some starting points (feel welcome to ignore these and follow a completely different approach): To decide which points to keep, given the intuition of Figure 1, you may want to keep points that are "close" to points from another class; and you may not want to keep points that are redundant, in the sense that you already have points from the same class near them. There are also several ways you can use to go back from the LP relaxation to the integral solution; for example, you simply round; or you could interpret the real values as probabilities, and keep each data set by flipping a coin with this probability.

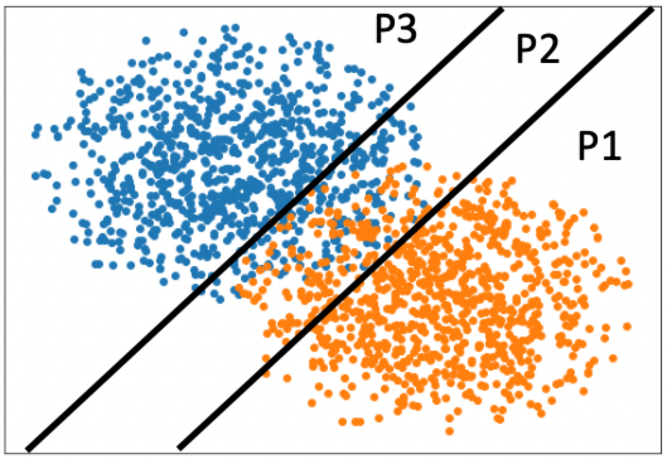

Figure 1: Toy example for sample selection.

1 Data set

We will use the same two data sets as in Project 1:

1. A synthetic dataset, with two classes, where each class is created through a Gaussian distribution. The means of the classes will be $\mu_1 = [-1, 1]$ and $\mu_2 = [1, -1]$, and they will share the same covariance matrix

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{1}$$

   Please generate 6000 training data points and 1000 test points from each class (in total, you should have 12000 training points and 2000 test points).

2. We will use the MNIST database of handwritten digits for the training and testing of the classifiers. The dataset is composed of 28x28 black and white images. There are 60000 training examples and 10000 test examples in this dataset. You can download the training and test images by following the link `http://yann.lecun.com/exdb/mnist/`. If you would like to use csv files, you can follow the link `https://www.kaggle.com/oddrationale/mnist-in-csv`. In this project, you won't use the whole MNIST dataset. You will train the linear classifier to distinguish between only the digits 1 and 7.

2 ILP and LP implementation

- `ILP`: this is where you will implement your ILP that determines which points to keep for the classification task. It takes the following inputs.
  - `self`: this is a reference to the MyClassifier object that is invoking the method.

  - `training_set`: $N \times M$, where M is the number of features in data vectors and N is the number of data points in the training set.
  - Output of this function should be the data set that will be used during training which is a matrix of dimensions $N_{train} \times M$. The output is a subset of the input training set, thus $N_{train} \leqslant N$.

- `LP`: this is where you will solve the ILP through LP relaxation and then construct an integral out of the real solution. It takes the same inputs as the ILP method.

- `train`: this is where you will implement the code that trains your classifier. It takes the following inputs.

  - `self`: this is a reference to the MyClassifier object that is invoking the method.
  - `train_data`: this is a matrix of dimensions $N_{train} \times M$, where M is the number of features in data vectors and N_{train} is the number of data points used for training. Each row of `train_data` corresponds to a data point.
  - `train_label`: this is a vector length N_{train}. Each element of `train_label` corresponds to a label for the corresponding row in `train_data`.
  - Output of this function should be the MyClassifier object which has the information of weights, bias, number of classes, number of features. You may use this object call the other functions as well.

  Although the MNIST dataset has 10 classes in total, we are expecting you to extract training and test data with labels corresponding to digits 1 and 7 and write a classifier that classifies the data either as 1 or as 7. Note that you still have to write your code in a generic way so that we are able to run your code with a different pair of classes e.g: 2 and 5.

- `f`: here you will implement the function $f(\cdot)$, which takes in a scalar value and outputs an estimated class $\hat{s}$. It takes as input two variables:

  - `self`: this is a reference to the MyClassifier object that is invoking the method.
  - `input`: this is the input which corresponds to the function $g(y) = W^T y + w$.
  - Output is an estimated class.

- `test`: this method should be used to test the classifier on the input data. The method takes as input the test dataset to be classified. The output of the function should be a vector that contains the classification decisions. Note that we are expecting you to classify the data as 1 and 7 on MNIST.

3 Report

We expect a joint report of up to 4 pages (approximately 2 pages for Part I and 2 pages for Part II).

For Part II, please explain the rationale of the objective function and the constraints of the ILP; also explain how you convert the real solution of the associated relaxed LP to an integral solution. Plot the performance of the classifier for different parameters of your design, and identify trade-offs between classification performance and number data samples used. In particular, report how many data samples your algorithm requires to achieve the following values of accuracy: 50%, 65%, 80%, and 95%. Discuss your findings.
Extra 3 points: compare the performance of your online Part I algorithm with the performance of the genie-aided centralized algorithm and discuss the differences.

4 Grading

- The second part of the project is worth 15 points (and you can have also 3 extra points for comparing part I and part II results).
- Provide the code that you implement in the most self-sufficient manner, so that you would expect it to run on virtually any machine with Python.
- We may use a different dataset when grading, so make sure your implementation of the classifier is generic, i.e., it takes inputs of any number of features.
- We may also use a different pair of classes (other than classes 1 and 7) to run your code so you need to have a flexible code.
- You need to submit a folder named `'group_{groupnumber}'` (`group_5` for the group with name group 5) on Gradescope. Inside this folder there should be a file named `MyClassifier_{groupnumber}.py` (`MyClassifier_5.py` for group 5). Also you should include your report in this folder.