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

**EE236A: Linear Programming**

Project Part I - Compression of Data for Learning
Due: Tuesday, November 16th, 2021

**Project Description:**
In the first part of the project, you will explore the "usefulness" of data points for training a linear classifier. This is an open-ended project, which means there are no "right" answers.
The first part of the project is due on **Nov 16th**.

# 1 Background

A classification task deals with vectors that have labels associated with them, and attempts to determine a label of a vector given the entries of that vector. More specifically, let $y \in \mathbb{R}^m$ be a vector of $m$ entries (we will refer to the entries as *features*), and let $s \in \{+1, -1\}$ be its label. A classification task is comprised of computing $\hat{s} = f(g(y))$ which is an estimate of $s$. The classification tasks $f(g(y))$ can be generally decomposed into:

- $g(y)$: a transformation function which extracts information from the input vector $y$.
- $f(x)$: a decision function, which takes the output of $g(y)$ and makes the final decision regarding the label estimate of $y$.

Examples:

- A classifier can classify images whether they are images of "cats" or "dogs". In that case, a vector $y$ could correspond to a concatenation of the pixel values for the image, and $s = 1(-1)$ to label the vector as "dog" ("cat").
- A classifier can classify voices whether they belong to a "man" or a "woman". In that case, a vector $y$ could correspond to a the voice signal, and $s = 1(-1)$ to label the vector as "man" ("woman").

A linear classifier is a classifier for which the function $g(y)$ is linear, that is, it can be written as $g(y) = W^T y + w$, where $W \in \mathbb{R}^{M \times L}$ and $w \in \mathbb{R}^L$.

**Assessing Classifiers Performance.** A good classifier is a classifier which is able to correctly classify "many" input vectors. Different linear classifiers correspond to different choices of $W$, $w$ and $f(\cdot)$. One way to measure the quality of a classifier is through the percentage of correctly classified points. Specifically, let $\mathcal{Y} \subseteq \mathbb{R}^M$ be a collection of feature vectors with $|\mathcal{Y}| = N$, and for each $y_i \in \mathcal{Y}$, let $s_i$ be its label. Let $\mathcal{S}$ be the corresponding set of labels. Then we can define the percentage of correct classification as

$$P(\mathcal{Y}, \mathcal{S}) = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}\left(s_i = f(W^T y_i + w)\right)$$

where $\mathbb{1}(x)$ is the indicator function (is equal to 1 when the event $x$ is true, and 0 otherwise).

**How to find the right classifier - Training a Classifier.** How do we find a good classifier, i.e., how do we find $W$, $w$ and $f(\cdot)$? The prevalent method of finding good classifiers for a particular class of inputs is the idea of "supervised learning" – for the remaining discussions on supervised learning and the project details, we will focus only on linear classifiers.

In supervised learning, the classifier designer is given a relatively large set of inputs (called *training data*) and their corresponding labels. We denote these sets as $\mathcal{Y}^{\text{Train}}$ and $\mathcal{S}^{\text{Train}}$ respectively. The classifier is then trained in some way to determine the best parameters such that these training data are correctly classified. More specifically, a particular decision function $f(\cdot)$ is chosen. Then, the parameters $W$ and $w$ are chosen so that the quantity $P(\mathcal{Y}^{\text{Train}}, \mathcal{S}^{\text{Train}})$ is sufficiently high. The ultimate goal of the classifier is clearly not to correctly classify $\mathcal{Y}^{\text{Train}}$, since for these points the correct labels are already known. However, the premise is that if the sets $(\mathcal{Y}^{\text{Train}}, \mathcal{S}^{\text{Train}})$ are good representatives of the type of input vectors that are going to be classified, the classifier would still perform closely to $P(\mathcal{Y}^{\text{Train}}, \mathcal{S}^{\text{Train}})$.

**Example - Finding a Separating Hyperplane.** Consider the following decision function

$$f(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{if } x \leqslant 0 \end{cases}$$

Now, let $W \in \mathbb{R}^M$ and $w \in \mathbb{R}$, that is, $g(y) = W^T y + w$. In that case, a point $y$ is labeled $+1$ if $W^T y + w > 0$ and is labeled $-1$ otherwise. Now, given a training data set ($\mathcal{Y}^{\text{Train}}$,$\mathcal{S}^{\text{Train}}$), finding a good classifier (i.e., $W$ and $w$) corresponds to finding a suitable hyperplane that separates the points with opposite labels. This is exactly similar to the example of finding a separating hyperplane discussed in lecture, which can be solved efficiently using a linear programming approach. This is one example where linear programming can be used to train classifiers.

## 2 Project Goal and Details

In this project, we consider a central node that wishes to train a classifier by collecting (labeled) data points from distributed sensors, as depicted in Fig. 1. We assume that the sensors communicate

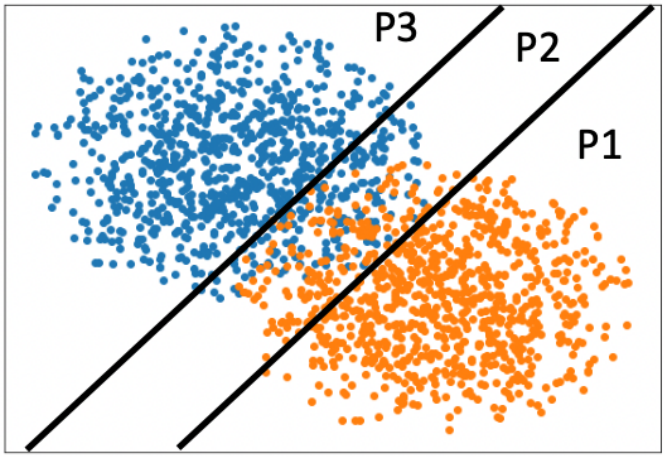


Figure 1: Toy example for sample selection.

with the central node over channels that are communication constrained; that is, there is a cost associated with each data point transmission, and thus we want to minimize the number of data points that the sensors transmit. Our main observation is that, some data points are more helpful than others, in designing the classifier. For example, consider the toy example in Fig. 1, where we have two classes, the orange and the blue points. One could perhaps argue that the points in P1 and P3 would be less useful than the points in P2 in helping to determine where is the classification boundary.

This idea is related to the notion of "support vectors", which are defined as the data points that are close to hyperplanes and influence a classifier's orientation. These points are used in the SVM algorithm to maximize the margin of the classifier.

The challenge in this project, is that, each sensor needs to decide in an "online" manner, whether to send a specific data point it has observed or not, without knowledge of the remaining points in the training dataset.

In this project, we will also assume that the central node can communicate with no constraints to the sensors, that is, although the uplink channels from the sensors to the central node are communication constrained, the downlink channel has no such constraints.

One naive approach that we can use for this task is to select a set of data points randomly to train the classifier. This is a straightforward approach we can compare against but the question is can we do better? Thus, you need to design an algorithm that decides on which data samples to transmit from sensor nodes to the central node such that the classifier in the central node performs close to what it would have been using all the data samples.

## 3 Data set

We will use two data sets:

1. A synthetic dataset, with two classes, where each class is created through a Gaussian distri-

bution. The means of the classes will be $\mu_1 = [-1, 1]$ and $\mu_2 = [1, -1]$, and they will share the same covariance matrix

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \quad (1)$$

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.

## 4 Classifier Implementation

You should implement a binary classifier in the project and the implementation of the classifier should go into the file `MyClassifier_{groupnumber}.py`. It should be an implementation of a class with four methods:

- `sample_selection`: this is where you will implement your algorithm that determines whether a data sample is crucial for the classification task or not. It takes the following inputs.
  - `self`: this is a reference to the MyClassifier object that is invoking the method.
  - `training_sample`: this is a vector with size $M$ where $M$ is the number of features in a single training sample.
  - MyClassifier object keeps the training data set which is initialized as an empty list. If the algorithm determines that the input data sample is important enough to use in the training, this data sample is added to the data set. Output of this function should be the MyClassifier object that keeps the updated data set. You may use this object call the other functions as well.
- `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 an $L$-dimensional vector 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 vector to $f(\cdot)$, which corresponds to the function $g(y) = W^T y + w \in \mathbb{R}^L$.
  - 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.

When implementing your algorithm, make sure to make it a generic implementation that takes inputs of any number of features. Although you will mainly try your classifier on the particular 2 classes of data set provided in this project (with 784 features and classes corresponding to 1 and 7), we may test your classifier on a different data set with different parameters.

## 5 Report

Beyond the code, we also expect a report of up to 2 pages, that describes in a complete way what is the rationale you used to decide which samples were sent as well as the specific algorithm description. We would like you to compare the performance of the classifier you trained, against the performance of the classifier that uses the full training dataset, and report the savings in terms of number of training samples. You should also compare the performance of your classifier against the performance of the classifier that uses a set of randomly selected data points (for example, you can select 30% of the data points randomly and train the classifier). You should also plot the performance of the classifier as a function of the number of data samples used for training it. Moreover, you need to report how many data samples your algorithm requires to achieve the following values of accuracy: $50\%, 65\%, 80\%, 95\%$ (in case it is able to achieve it).

## 6 Grading

- This part of the project is worth 15 points.
- 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). There should be a csv file named `weights_{groupnumber}.csv` (e.g. `weights_5.csv`) which contains a 1 dimensional array with 784 elements i.e. with shape (784,) and `bias_{groupname}.csv` (e.g. `bias_5.csv`) which contains a scalar. Also you should include your report in this folder.