Fall 2020
EE 236A
Christina Fragouli
Mine Dogan and Kaan Ozkara

EE236A: Linear Programming

Project I
Due: Thursday, November 5th, 2020

Project Description:
In this project, you will implement a robust classifier that can operate if some of the inputs are missing. This is an open-ended project, which means there are no “right” answers.
The deadline for submitting the project is on **Nov 3rd**.

Project I - Building a Robust Linear Classifier

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 is a 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, you would like to take a linear classification approach to create a multi-class classifier. In multi-class classification, the data input y has a label that takes one of K values. Therefore, the label estimate of your classifier (i.e., the output of the function $f(\cdot)$) can take any value from $1, \cdots, K$. Your goal is therefore to design 1) the function $f(\cdot)$, and 2) the parameters $W \in \mathbb{R}^{M \times L}$ and $w \in \mathbb{R}^{L}$.

Imagine that you have implemented a classifier, and that this classifier is going to be used over a distributed system (for example, a group of drones that perform cooperative sensing). In such a system, perhaps a different drone observes each of the features that form the input to the classifier (eg. takes a photo from a different angle), and all the observed features are send to a central node that has the classifier and will take the classification decision. However, because the network topology may change and some of the drones may be out of range, or perhaps the communication channels fail, it can happen that some of the input features could be missing at the test time. As an example, imagine that you trained the classifier based on training dataset that has two features, however, it is possible that you now receive some inputs with only one feature at the test time.

You would like to build a classifier that is robust against these types of errors. Namely, your classifier should be robust against errors that "erase" some of the input features at the test time. In particular, assume that in the test dataset every input feature is erased independently from others with probability p. You can assume that p is known during the training time. Therefore, you should implement the classifier to benefit from this knowledge. Note that by erasure it is meant setting pixel values to 0. You are expected to train by having $p = 0.6$, but in the testing stage you will test your model with 3 different p values: $0.4, 0.6, 0.8$.

You will take a supervised learning approach in designing your classifier. Once your classifier is trained, you should be able to assess its performance over another set of data inputs that is different from the set you used for training. The data sets used for training and testing are respectively denoted as $(\mathcal{Y}^{\text{Train}}, \mathcal{S}^{\text{Train}})$ and $(\mathcal{Y}^{\text{Test}}, \mathcal{S}^{\text{Test}})$. These data sets are discussed in detail in the next section.

3 Data set

We will use the MNIST database of handwritten digits for the training and testing of the classifiers. The dataset 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`.

4 Classifier Implementation

You should implement a **One-vs-One** 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 three methods:

- `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 classifier object that is invoking the method.
  - `p`: Erasure probability p.
  - `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 functions should be a MyClassifier object which has the information of weights, bias, number of classes, number of features. You may use this object to call the other functions as well.

  During training you will use $p = 0.6$, but you need to write your code in a generic way so we can train using different values of p.

  Although 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. So you need to have a subset of MNIST dataset; training and test data you are using should consist of only data with labels corresponding to 1 and 7. You need to train your classifier such that it 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 different number of classes e.g: 10.

- `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 classifier 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.

- `TestCorrupted`: this method should be used to test input data on the classifier with missing features. The method takes as input the erasure probability p, and a test dataset to be classified where for each datapoint, every feature is erased with probability p[1]. The function

[1] For grading purposes, the erasing of features should be done inside the function, i.e., the method takes a full test vectors as an input, erase some of the features, then classify.

would then attempt to classify the input vectors using only the features that are not missing. 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 classifier, make sure to make it a generic implementation that takes inputs of any number of features, and any number of classes. 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 Grading

- This project is worth 15 points.

- The report should include a plot for the classification accuracy (number of correctly classified test data points/total number of test datapoints) at the test time versus the erasure probability p. Use the values $p = 0.4, 0.6$ and 0.8. For each test datapoint, erase every feature with probability p. After updating the test dataset, the function would then compute the classification decisions. Calculate the percentage of correct classification. Do this experiment 10 times and report the average classification accuracy.

- For $p = 0.4, 0.6$ and 0.8, plot a histogram that displays the number of input features (out of 784) that got erased, over 10 trials.

- The report should be at most 3 pages, and you need to clearly describe your approach in solving the problem.

- Use the template file of the project to fill in the implementation of your classifier. As discussed above, you should implement additional methods to emulate the effect of errors. 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.

- For the purposes of grading, we will check your classifiers locally after submission. 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, and any number of classes.

- 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. We need to be able to use the submitted weights and bias for our own test data otherwise you will not be able to compete for bonus points. Also you should include your report in this folder.

Please consider start working on your project early so that you have sufficient time. Good luck.