Introduction to Machine Learning - Exercise 1

Due Date: November 17th 22:00, 2019

Yosi Shrem and Joseph Keshet

November 6, 2019

1 ERM

1.1

As mentioned in the class, a learning algorithm receives as input a training set S sampled from an unknown distribution $\mathcal{D}$ and labeled by some target function f. Since the learner does not know what $\mathcal{D}$ and f are, we use a training set of examples, which acts as a snapshot of the world that is available to the learner. In ERM we would like to find a solution that works well on that data.

An axis aligned classifier in the plane is a classifier that assigns the value 1 to a point if and only if it is inside a certain rectangle. Formally, given real numbers $a_1 \leq b_1, a_2 \leq b_2$, define the classifier $h_{(a_1,b_1,a_2,b_2)}$ by

$$h_{(a_1,b_1,a_2,b_2)}(x_1,x_2) = \begin{cases} 1 & \text{if } a_1 \leq x_1 \leq b_1 \textit{ and } a_2 \leq x2 \leq b_2 \\ 0 & \text{otherwise} \end{cases}$$

Let A be the algorithm that returns the smallest rectangle enclosing all positive examples in the training set. Explain whether A is an ERM or not.

1.2

Let $\mathcal{H}$ be the hypothesis space of binary classifiers over a domain $\mathcal{X}$. Let $\mathcal{D}$ be an unknown distribution over $\mathcal{X}$, and let f be the target hypothesis in $\mathcal{H}$. Denote $h \in \mathcal{H}$.
Let us define the *true error* of h as,

$$L_{\mathcal{D}}(h) = \mathbb{P}_{x\sim\mathcal{D}}[h(x) \neq f(x)]$$

Let us define the *empirical error* of h over the training set S as,

$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{[h(x)\neq f(x)]}$$

where m is the number of training examples.
Show that the expected value of $L_S(h)$ over the choice of S equals $L_{\mathcal{D}}(h)$, namely,

$$\mathbb{E}_{S\sim\mathcal{D}}\big[L_s(h)\big] = L_{\mathcal{D}}(h)$$

2 Sound Compression

In this part of the exercise we will use the k-means algorithm for sound compression, i.e. you should implement the k-means algorithm on the **amplitude values** and then replace each value by its centroid.
You should implement the k-means algorithm as described in class (slide no. 40 in recitation 1 presentation). You will train your algorithm and report results using the `sample.wav` as shown in Figure 1. For visualization, you can use Praat (`http://www.fon.hum.uva.nl/praat/`)

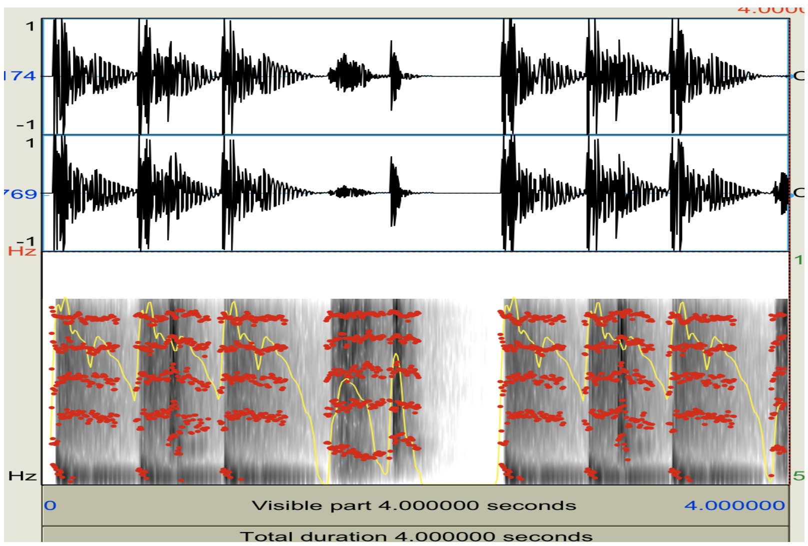

Figure 1: sample.wav

The centroids initialization will be provided to you in a text file which will be received as an argument to your program. The run command to your program should be:

```
$ python ex_1.py <wav file> <centroinds_init>
For example:
$ python ex_1.py sample.wav cents1.txt
```

The following snippet contains commands for reading and writing sound files as well as reading the centroids initialization:

```
sample,centroids = sys.argv[1],sys.argv[2]
fs, y = scipy.io.wavfile.read(sample) #reading
x=np.array(y.copy())
centroids=np.loadtxt(centroids)

#your code goes here#
...
...
scipy.io.wavfile.write("compressed.wav", fs, np.array(new_values, dtype=np.int16))#saving
```

You should get something similar results to the following,

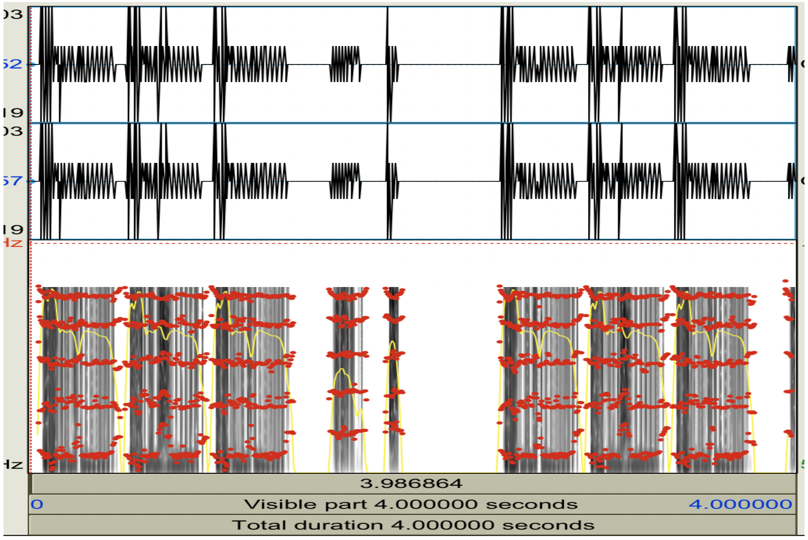

Figure 2: K=5

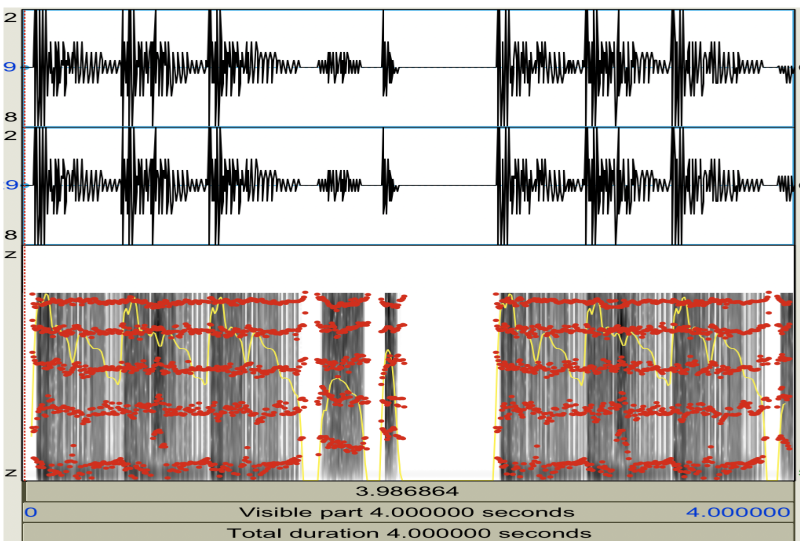

Figure 3: K=10

Reproducibility. Originally, the initial centroids in k-means are randomly generated. For reproducible purposes, we provided you with `cents1.txt` and `cents3.txt` for centroid initialization, you should use it and compare your output with `output1.txt` and `output3.txt`! Please note that given these pre-defined values, your sequence of centroid updates should be deterministic and not random in any way.

Note: in case when 2 centroids are evenly close to a certain point, the one with the lower index

"wins".
Your code should run for 30 iterations or until convergence. Your program should create `output.txt`, consisting of your centroids after each centroid update step as follows: For example, when using `cents3.txt`:

```
[iter 0]:[-4189. -4171.],[ 4. -2.],[3. 2.],[4. 3.],[4543. 4517.]
[iter 1]:[-7645. -7603.],[-494. -718.],[-243.   24.],[780. 802.],[7851. 7858.]

...
[iter 18]:[-24823. -24965.],[-7598. -7574.],[ 10. -11.],[7645. 7638.],[24409. 24914.]
[iter 19]:[-24823. -24965.],[-7598. -7574.],[ 10. -11.],[7645. 7638.],[24409. 24914.]
```

```
The full requested output is at output3.txt
use the following print-line to match outputs:
   print(f"[iter {iter}]:{','.join([str(i) for i in new_z])}")
```

As you can see the algorithm converged and stopped(same centroids). **For consistency, after each centroids update, use the** *round*() **function on each dimension**. **We define convergence when the centroids don't update.** You can assume both the sound and the centroids initialization files will be at the same working directory as your running file.

The `sample.wav` was taken from here : `https://www.youtube.com/watch?v=7vQ83lz0528` (Lucille Crew - Something).

3 What to submit?

You should submit the following files:

- A `txt` file, named `details.txt` with your name and ID.
- A `PDF` file named `ex_1.pdf` with your answers to 1.1 and 1.2.
- Python 3.6 code for question 2. Your main function should reside in a file called `ex_1.py`. The main function prints the centroids as explained above.
- A txt file named `output2.txt` - Your program's output when using the `cents2.txt` for initialization.
- A `PDF` report including the following plots: (i)The average loss value (i.e. the distance between each point to its closest centroid) as a function of the iterations(you can stop at 10) for $k = 2, 4, 8, 16$. Your report should include the implementation details(number of iterations, how you chose your centroids, did you run multiple run each time?, etc.)
- Part of your grade will consist of automatic checks using the `Submit` system. Make sure your output matches the expected output, check your mail after submitting.
- **Note**: your code should load n sound and a centroids initialization files located in the same folder as your main script. Make sure you use relative paths.

  Overall : `ex_1.py, ex_1.pdf, details.txt, output2.txt` and `report.pdf`