PYTHON IMPLEMENTATION OF ALGORITHM DESCRIBED IN ESTIMATING THE UNSEEN: IMPROVED ESTIMATORS FOR ENTROPY AND OTHER PROPERTIES

NEEL PARATKAR (111483570), ROHAN KARHADKAR (111406429), SHARAD SHRIDHAR (111492675), CHAITANYA KALANTRI (111446728)

Project Link: https://github.com/rkya/EstimatingTheUnseen

1 Introduction

There are many methods to find the relationship among the data. For instance, when finding the distribution of random sample, empirical methods may not always be efficient. As the distribution can be linear, non-linear or may not even be distributed in any definable pattern.

And in real world datasets. The data is highly random and can't always be categorized into uniform distributions. Hence, in the paper, we introduce a different entropy function, which is used to find the relationship among the data. It is more robust to find the "unseen" patterns within the data. We can estimate the shape or the histogram of the unseen portion of the data. And now, given such a reconstruction, one can any property of the distribution which only depends on the shape/histogram; such properties are termed symmetric and include entropy and support size.

In the paper, the algorithm proposed is based on the linear programming. It does not even have an objective function and simply defines a feasible region.

2 Previous Work

There are several proposals about estimating the relationship among the elements of the data. For instance, both R.A. Fisher and Alan Turing during the 1940's. Fisher was presented with data on butterflies collected over a 2-year expedition in Malaysia and sought to estimate the number of new species that would be discovered if a second 2 year expedition were conducted [8]. (His answer was "≈ 75."). In contrast to this approach of trying to estimate the "shape/histogram" of a distribution, there has been nearly a century of work proposing and analyzing estimators for particular properties of distributions.

3 Estimating the Unseen

Given the fingerprint F of a sample of size k, drawn from a distribution with histogram h, our high-level approach is to find a histogram h' that has the property that if one were to take k independent draws from a distribution with histogram h' , the fingerprint of the resulting sample would be similar to the observed fingerprint F. The hope is then that h and h' will be similar, and, in particular, have similar entropies, support sizes, etc.

To explain the above in details. Let's go over the pseudo code:
Basically, to compute the unseen, we first come up with any plausible histogram and then reach the optimal plausible histogram.
The functions in the algorithm:

1. Compute the unseen
2. Come up with any plausible histogram
3. Get the optimal plausible histogram

The input given to the function computing the optimal plausible histogram contains the Fingerprint vector and error pattern. The output contains pairs of mesh of points and a plausible histogram.

4 Our Implementation of Algorithm

To estimate the entropy of a sample distribution (which is a symmetric property), we try and estimate the shape/histogram of the unseen portion of the distribution. Some important definitions to be used in the algorithm are introduced below:

- **F: Fingerprint of a sample** – the histogram of the histogram of the sample
- **π: Symmetric distribution property** – a function/property that depends only on the histogram of the distribution and is invariant to permutations of the sample
- **R(p1, p2)**: Relative earth-mover distance - the minimum over all schemes of moving the probability mass of the first histogram (of p1) to yield the second histogram (of p2)
- **Poi(λ, j):** Poisson distribution of expectation λ - probability that a random variable with distribution Poi(λ) takes value .

- **Algorithm Flow**

The main essence of the algorithm is the use of a pair of linear programs, one after the other.

1. The first linear program returns a histogram (h') that minimizes the distance between its expected fingerprint and observer fingerprint. After this, the second linear program will find the histogram (h'') of minimal support size, subject to the constraint that the distance between its expected fingerprint, and the observed fingerprint, is not much worse than that of the histogram found by the first linear program
2. Before we run the linear program solver, first split the given fingerprint into the dense (or the hard portion) and sparse (easy) portion. The dense portion will be used for solving the linear program and for the sparse portion, we will use the empirical histogram.
3. We then use the dense portion as input to the first linear program. The output of this linear program is a histogram vector (h1',……,hl'), and an optimal objective function value that we then use, in addition to the fingerprint, and an error parameter to solve the second linear problem.
4. After the second linear program, we obtain a solution histogram vector (h1',…..,hl'), which we will try to optimize, so as to make sure the distance between the expected and observed fingerprint is not much worse than that of the histogram obtained from the first linear program. This linear program solution is then combined with the empirical portion of the histogram.
5. We now use this result to calculate the entropy of the distribution and compare the results with the entropy obtained from other methods (naïve, empirical, etc.)

- **Linear Programming**

Simply put, a linear programming problem may be defined as the problem of maximizing or minimizing a linear function subject to linear constraints. The constraints may be equalities or inequalities. For programming languages such as Python, we have libraries that make use of 'solvers' for linear programming. Solvers, in this case refer to the basis-exchange algorithms that are used to compute the results. We looked at three libraries that could be used with Python to solve the linear program. SciPy has an optimized solver that minimizes a linear objective function to linear equality/inequality

Figure 1: Image Source: https://theory.stanford.edu/~valiant/papers/unseenJournal.pdf

constraints. SciPy can make use of two solvers: 'Simplex' and 'Interior-point'. We used the 'Interior-point' solver in our implementation.

Python also has a package, 'Pulp' which provides us with an interface to several different solvers such as 'CBC'. This is quite useful if we want to use different types of solvers, especially ones that are licensed and aren't usually bundled with other scientific libraries provided by Python.

The 'Mosek' API provides an object-oriented interface to its various optimizers (linear, conic quadratic, convex quadratic, etc.). This works like an external solver which can be used within other scientific libraries such as SciPy.

We tested out the usage of two different solver, namely: **Simplex** and **Interior-point.**

1. **Simplex** – this algorithm solves LP problems by constructing a feasible solution at a vertex of the polytope (any geometric object with flat sides) and then walking along a path on the edges of the polytope to vertices with non-decreasing values of the objective function until an optimum is reached for sure.
2. **Interior-point** - In contrast to the simplex algorithm, which finds an optimal solution by traversing the edges between vertices on a polyhedral set, interior-point methods move through the interior of the feasible region.

Algorithm 1. ESTIMATE UNSEEN

Input: Fingerprint $\mathcal{F} = \mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$, derived from a sample of size n, vector $x = x_1, \ldots, x_\ell$ with $0 < x_i \leq 1$, and error parameter $\delta > 0$.
Output: List of pairs $(y_1, h'_{y_1}), (y_2, h'_{y_2}), \ldots$, with $y_i \in (0, 1]$, and $h'_{y_i} \geq 0$.

- Initialize the output list of pairs to be empty, and initialize a vector $\mathcal{F}'$ to be equal to $\mathcal{F}$.
- For $i = 1$ to n,
  - If $\sum_{j \in \{i - \lceil \sqrt{i} \rceil, \ldots, i + \lceil \sqrt{i} \rceil\}} \mathcal{F}_j \leq 2\sqrt{i}$ [i.e. if the fingerprint is "sparse" at index i]
    Set $\mathcal{F}'_i = 0$, and append the pair $(i/n, \mathcal{F}_i)$ to the output list.[4]
- Let v_{opt} be the objective function value returned by running Linear Program 1 on input $\mathcal{F}', x$.
- Let h be the histogram returned by running Linear Program 2 on input $\mathcal{F}', x, v_{opt}, \delta$.
- For all i s.t. $h_i > 0$, append the pair (x_i, h_i) to the output list.

Linear Program 1. FIND PLAUSIBLE HISTOGRAM

Input: Fingerprint $\mathcal{F} = \mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$, derived from a sample of size n, vector $x = x_1, \ldots, x_\ell$ consisting of a fine mesh of points in the interval $(0, 1]$.
Output: vector $h' = h'_1, \ldots, h'_\ell$, and objective value $v_{opt} \in \mathbb{R}$.

Let $h'_1, \ldots, h'_\ell$ and v_{opt} be, respectively, the solution assignment, and corresponding objective function value of the solution of the following linear program, with variables $h'_1, \ldots, h'_\ell$:

$$\text{Minimize: } \sum_{i=1}^{m} \frac{1}{\sqrt{1+\mathcal{F}_i}} \left| \mathcal{F}_i - \sum_{j=1}^{\ell} h'_j \cdot poi(nx_j, i) \right|$$

$$\text{Subject to: } \sum_{j=1}^{\ell} x_j h'_j = \sum_i \mathcal{F}_i / n, \text{ and } \forall j, \ h'_j \geq 0.$$

Linear Program 2. FIND SIMPLEST PLAUSIBLE HISTOGRAM

Input: Fingerprint $\mathcal{F} = \mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$, derived from a sample of size n, vector $x = x_1, \ldots, x_\ell$ consisting of a fine mesh of points in the interval $(0, 1]$, optimal objective function value v_{opt} from Linear Program 1, and error parameter $\delta > 0$.
Output: vector $h' = h'_1, \ldots, h'_\ell$.
Let $h'_1, \ldots, h'_\ell$ be the solution assignment of the following linear program, with variables $h'_1, \ldots, h'_\ell$:

$$\text{Minimize: } \sum_{j=1}^{\ell} h'_j \quad \text{Subject to: } \sum_{i=1}^{m} \frac{1}{\sqrt{1+\mathcal{F}_i}} \left| \mathcal{F}_i - \sum_{j=1}^{\ell} h'_j \cdot poi(nx_j, i) \right| \leq v_{opt} + \delta,$$
$$\sum_{j=1}^{\ell} x_j h'_j = \sum_i \mathcal{F}_i / n, \text{ and } \forall j, \ h'_j \geq 0.$$

Figure 1: Estimate Unseen Function Pseudo code

5 Empirical Results

The authors of the paper implemented the algorithm and tested it on a number of different cases. The Estimator mentioned in the paper was compared against five different entropy estimators.

1. **The "Naïve" Estimator**: the entropy of the empirical distribution given by a fingerprint F, calculated over a sample size of n is given by , $H^n(F) := -\sum_i F_i * \frac{i}{n} |\log_2 \frac{i}{n}|$
2. **The Miller-Madow Corrected Estimator**: The Naïve Estimator is corrected to account for second derivative of the logarithm function given by,
$$H^{MM}(F) := k.H^n - \frac{n-1}{n} \sum_{j=1} H^n.(F^j)$$
3. **The jackknifed naïve estimator:** It is given by:
$$H^{JK}(F) = k.H^n(F) - \frac{n-1}{n} \sum_j H^n(F^{-j})$$
4. **The Coverage Adjusted Estimator(CAE):** It was designed to apply to settings in which there is a significant component distribution

Figure 1: Image Source: https://theory.stanford.edu/~valiant/papers/unseenJournal.pdf

5. **The Best Upper Bound Estimator:** It is obtained by searching for a minimax linear estimator, with respect to a certain error metric.

For our implementation in **Python,** we ran experiments to estimate the unseen entropy for k = 1000 and k=10000 for a wide range of n. For each combination of n and k we called the function to estimate the entropy 100 times and calculated the **Root Mean Square Error**. For the uniformly distributed sample space we plotted the graphs for the RMSE against different values of n. Below graphs show the results we got after running our experiments.

We also calculated the entropy by three of the above-mentioned estimators viz. Naïve Estimator, Miller-Madow corrected estimator and jackknifed naïve estimated. The results we got were quite similar to the what the authors received after running the experiments. As we can see from the graphs, the Unseen Estimator Algorithm works better than the three other estimators. The value of RMSE is high when $n < k$ and it drops drastically as n becomes equal to k. The RMSE is very low for n > k.

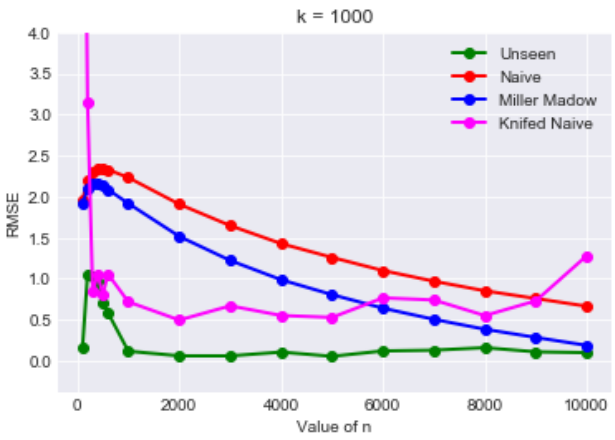

Figure 2: Entropy estimation for k=1000

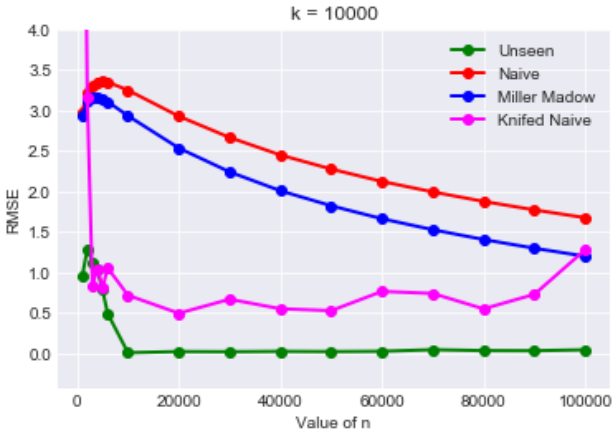

Figure 3: Entropy Estimation for k=10000

6 Conclusion

The aim of this project was to provide a non-matlab implementation of the algorithm described in the research paper. We have successfully implemented the mentioned algorithm in **Python** using SciPy library. We compared the running time of our implementation vs the matlab implemented that was available on authors' website. For a value of k = 1000 with n varying from 100 to 100000, for 100 iterations of each combination of n and k, the matlab code runs in *1min05secs,* while our code ran in *2min15 secs.* Similarly for k = 10000 with n varying from 1000 to 1000000, for 100 iterations of each combination of n and k, the matlab code runs in *1min35secs,* while our code ran in *2min19 secs.* The RMSE values of the estimated entropy using our algorithms presented similar trends to the ones described in the paper which points to a significantly good implementation of the code in python.

7 Future Scope

Although SciPy library provides a *linprog* function with solvers that work in considerably low amount of time, we plan to try out external solvers that can be used along with SciPy library's *linprog()* function. These include the PuLP library and Mosek Library mentoined in the previous sections. The program can

Figure 1: Image Source: https://theory.stanford.edu/~valiant/papers/unseenJournal.pdf

also be tested against several other distribution types such as ZipF distribution, Mixed Uniform distribution, Geometric Distribution. etc.

8 Project Link

https://github.com/rkya/EstimatingTheUnseen

9 References

[1]https://papers.nips.cc/paper/5170-estimating-the-unseen-improved-estimators-for-entropy-and-other-properties.pdf
[2]https://docs.scipy.org/doc/
[3]https://stackoverflow.com/questions/10697995/linear-programming-in-python
[4]https://theory.stanford.edu/~valiant/papers/unseenJournal.pdf
[5] https://theory.stanford.edu/~valiant/
[6] https://matplotlib.org/

Figure 1: Image Source: https://theory.stanford.edu/~valiant/papers/unseenJournal.pdf