Randomized Algorithms versus Deterministic Algorithms

Introduction:

Randomization introduces magic to the world of algorithms. They work their magic for ill-structured global optimization problems to find good solutions by sacrificing a guarantee of optimality. Randomness is idiosyncratic due to the uncertainty related to it. In today's world , randomization is an extremely important tool for the construction of algorithms for a wide range of applications.[5]

Background:

Powerful randomized algorithms were developed in number theory such as Legendre symbol and quadratic residue that lead to planting the seed of randomization in the computational field. Randomization in the area of algorithms drew attention in the late 70's when Rabin posted a note in his journal known as the Probabilistic Algorithms [1] in 1976 followed by Strassen and Solovay who posted a paper on the Monte Carlo algorithm in 1977 [2] and J.Gill who posted a paper on the computational complexity of the Turing machines [3].

In today's world, randomized algorithms are extensively used in cryptography with pseudo random number generators (The algorithm that generates the sequence of numbers whose properties are like that of the random numbers.) Spreadsheets and database programs also involve sorting data that is done by using Randomized Quick Sort.

The difference between deterministic algorithms, probabilistic algorithms and randomized algorithms must be mentioned. Deterministic algorithms as shown in Figure 1 has an input to an algorithm which gives the output. Figure 2 shows random numbers applied to probabilistic algorithm to give an output. In Figure 3, randomized algorithms have a randomized input to produce the output.

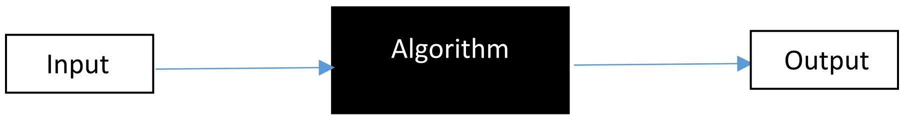

Figure 1: Deterministic Algorithms

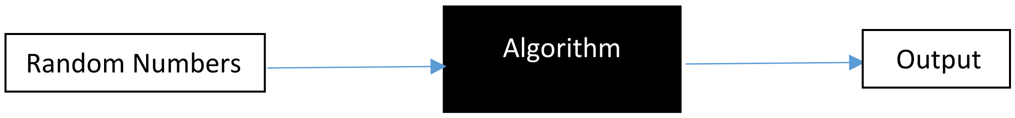

Figure 2: Probabilistic Algorithms

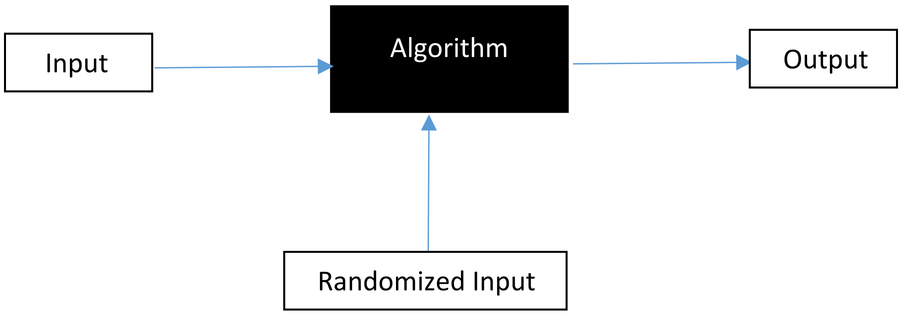

Figure 3: Randomized Algorithms

There are two major advantages to using randomization. One of the advantages is the increase in running time and the decrease in the space requirements while the other is the ease of understanding and implementing a random code while compared to the deterministic algorithms. An easy deterministic algorithm with a bad running time can be randomized to give a better running time by giving high-probability random inputs.

The reason this topic was chosen for my project was due to the uncertainty associated with inserting randomization to any algorithm. The feeling is like that of playing against an adversary and waiting for his next move. It is a tool that is emphasized on due to the power it possesses to modify an algorithm to give better results.

This project implements the algorithms for sorting and classification. Quick sort is implemented in the language Python and random forest along with K-nearest neighbor algorithm using the language R. The data set for classification was taken from the website Kaggle[4].

The agenda of the project is to show how randomness just helps improve the results of a deterministic algorithm.

Body:

Quicksort:

The Quicksort algorithm is a sorting algorithm that has a worst case of $O(n^2)$ time when implemented in the deterministically. The pivot is chosen as the first element or the last element in the input given and is sorted to find the ascending order of the sequence.

Randomization is introduced to the algorithm by choosing a random number in the input as a pivot and sorting. The expected running time of the randomized algorithm is O(n*logn). The programs for both have been implemented programs and the running time as the input keeps increasing reduces for the randomized implementation.

Quicksort:

Input Size	Time(seconds)
10	$1.33\text{x}10^{-5}$
10^2	0.00021
10^3	0.00272
10^4	0.06592
10^5	2.94210
10^6	322.98542

Table 1: Deterministic Running Time

Randomized Quicksort:

Input Size	Time(seconds)
10	$3.72\text{x}10^{-5}$
10^2	0.00069
10^3	0.00678
10^4	0.08213
10^5	1.1634
10^6	55.9843

Table 2: Randomized Running Time

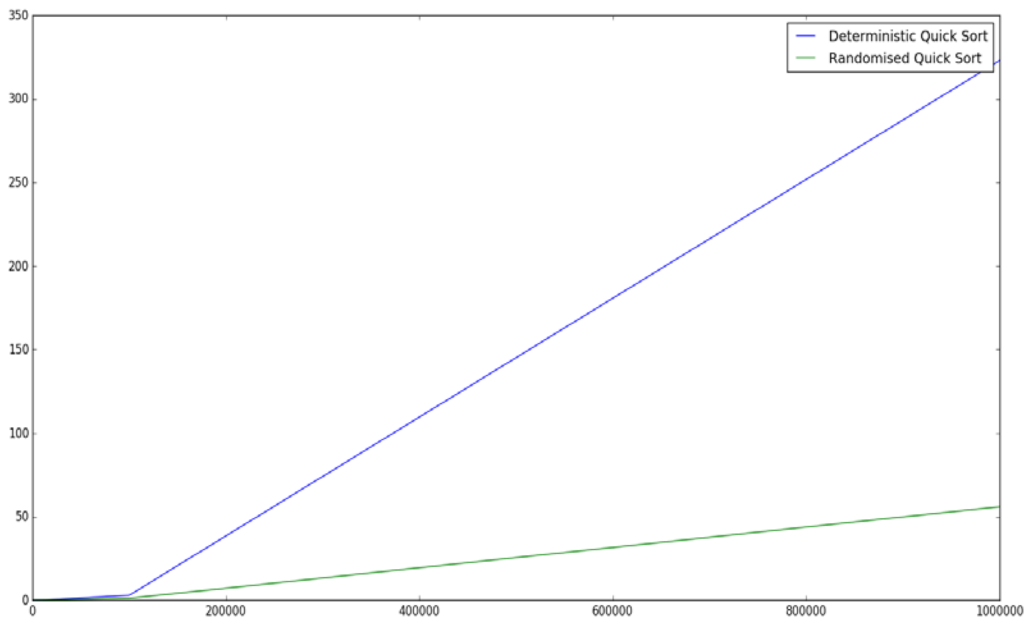

Figure 4: Running time of the Quicksort Algorithms

As we can see, the running time of the randomized algorithm is significantly smaller than the deterministic running time as the size of the input keeps increasing.

Probabilistic space consists of sequences of random pivot choices. The probability of choosing a number on the extremes for a bad split would be smaller than choosing any other number in the dataset due to basic mathematics. The probability of choosing a number among (n-2) elements in the input is easier than choosing one of the extremes. The splitting in the input given , even if it's a 9:1 split, T(n) where n is the size of the input is going to be n + T(n/90) + T(n/10) which results in O(n*logn) time whereas in deterministic Quick Sort. If the input given is a sorted list the algorithm is going to run in O(n^2) time which does not apply to the randomized version of the algorithm as it does not distinguish between inputs given to a program.

Random Forests:

The other randomization topic chosen was Random Forest and KNN algorithms. Both the algorithms are used to classify a dataset in machine learning.

The data set chosen classified the Selling Price of houses in regard with various factors such as the garage size, the locality surrounding the house, the size of the house, the number of bedrooms and a lot more factors.

The dataset used was initially cleaned for the missing values. The columns in the dataset with characters were factored and given numerical values.

The confusion matrix was generated for both the classifying algorithms to determine accuracy and error.

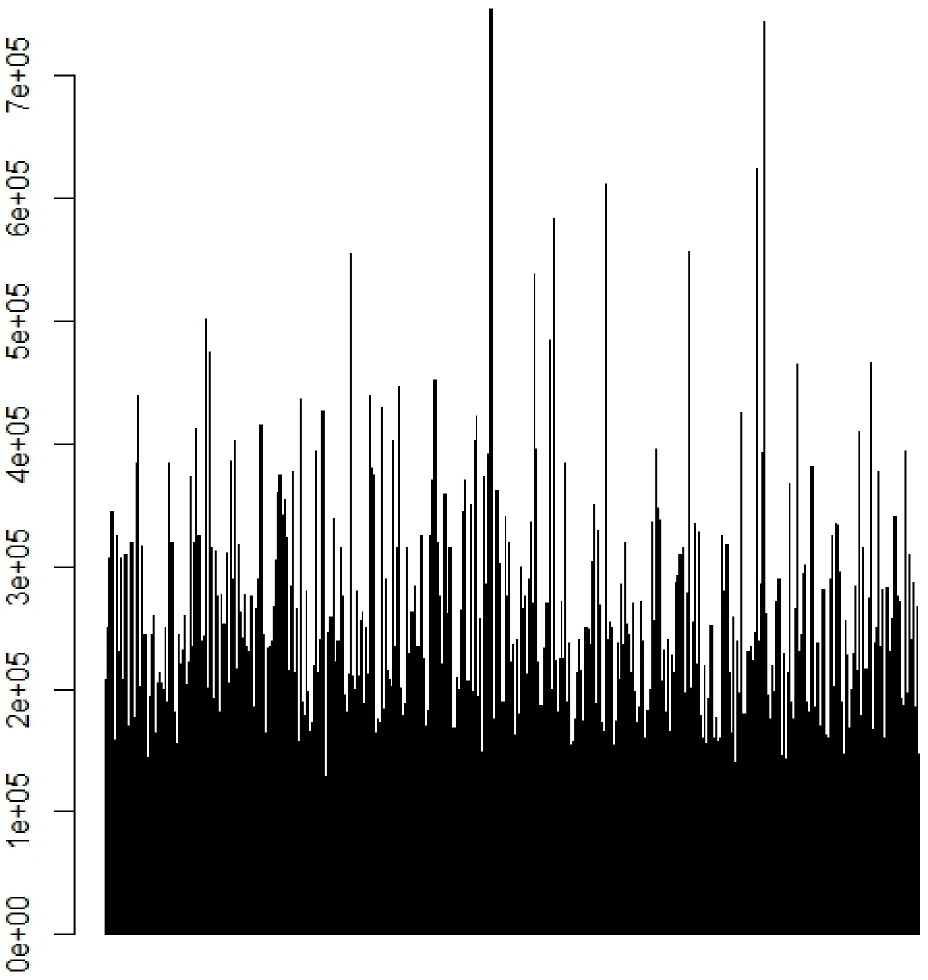

Figure 5: The Sales prices given in the dataset

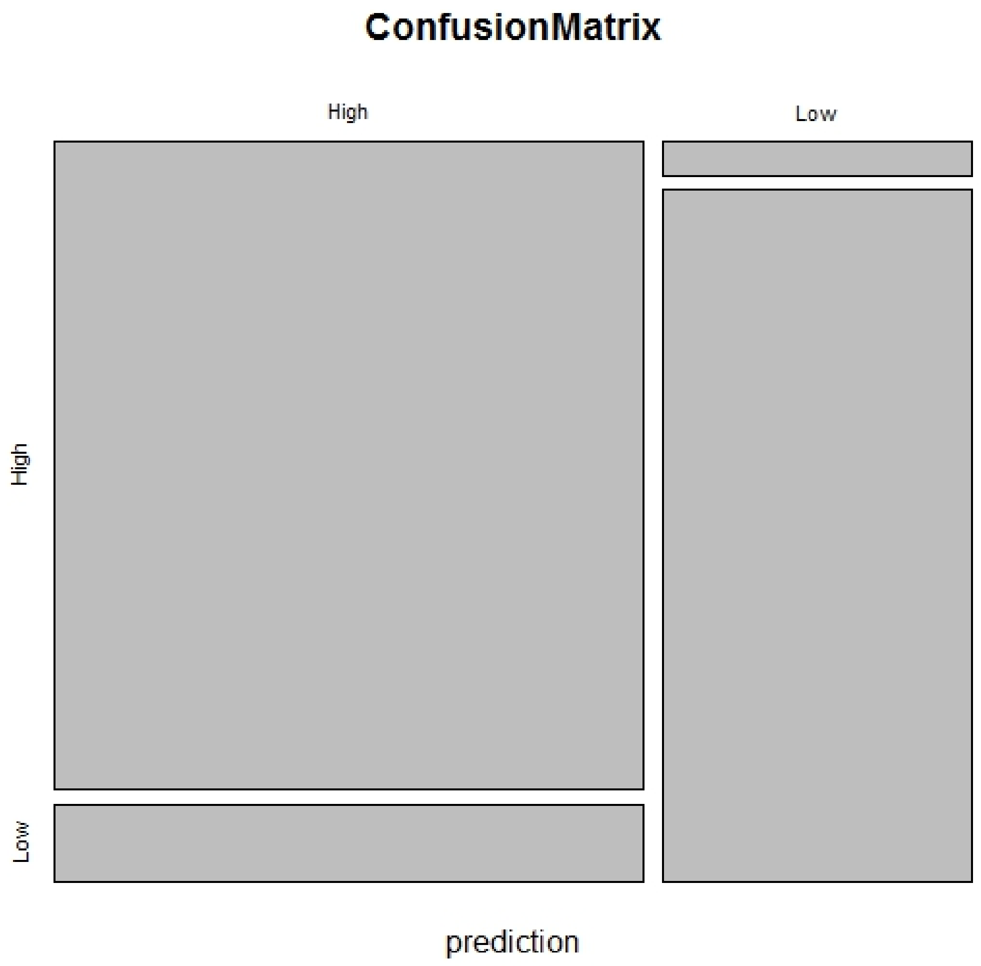

Figure 6: The Confusion Matrix for the Sales Price prediction using Random Forests.

The Sales Price was high if it was greater than the mean of the Sale prices calculated and lower otherwise.

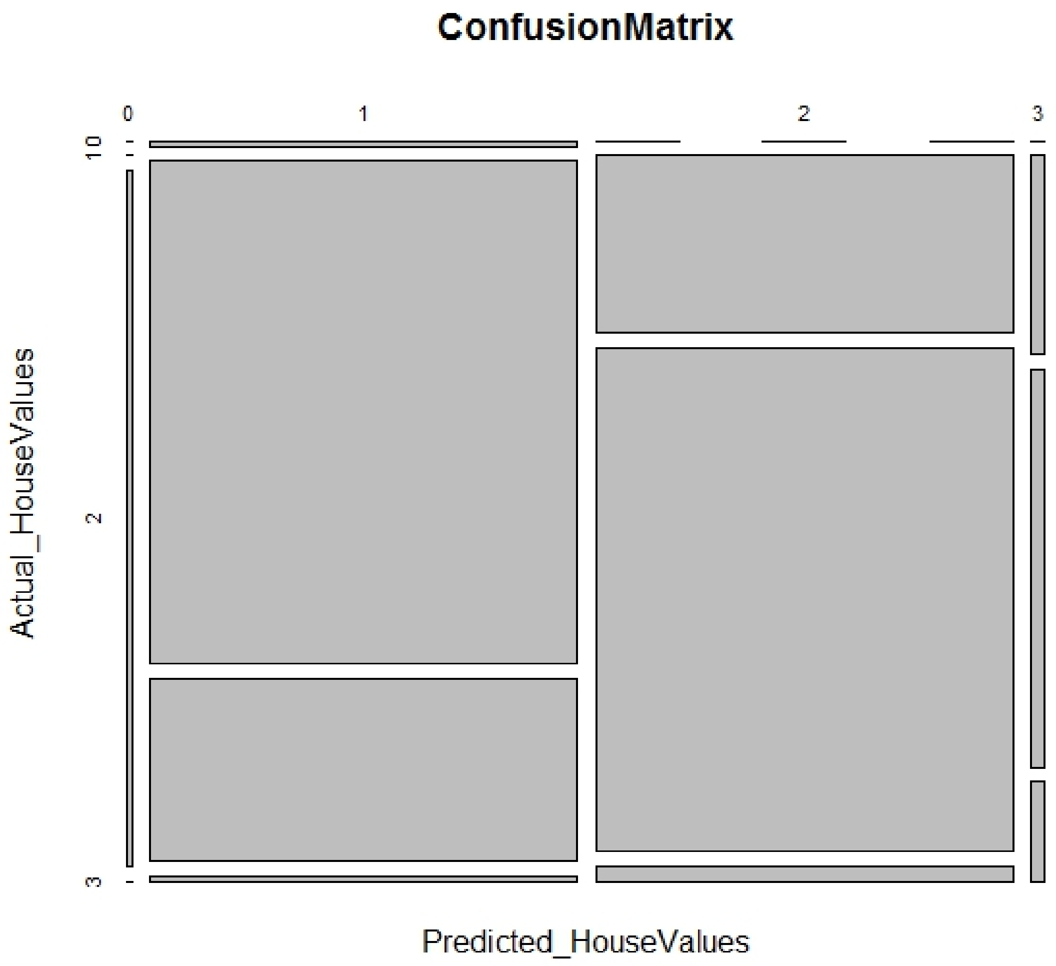

Figure 7: The Confusion Matrix for the Sales Price prediction using the KNN Algorithm.

The confusion matrix for KNN predicted prices equal to the mean prices and hence it classified it into three categories. Prices classified as 1 were lower than the mean, prices classified as 2 were higher than the mean and prices classified as 3 were equal to the mean.

The error rate and accuracy can be derived from the matrix.

Confusion Matrix - True Positive - A High Sales Price classified as High.

Confusion Matrix - False Negative - A Low Sales Price classified as High.

Confusion Matrix - False Positive - A Low Sales Price classified as Low.

Confusion Matrix - True Negative - A High Sales Price classified as Low.

Error = (False Positive + False Negative) / (True Positive + False Negative + False Positive + True Negative)

Accuracy = (True Positive + True Negative) / (True Positive + False Negative + False Positive + True Negative)

	Random Forests	KNN
Accuracy	91.8%	78.6%
Error	8.01%	18.3%

Table 3: Accuracy and error of the classifiers.

The results of Random forests are much greater than that of the KNN as seen by the table above. Random forests classify the data based on randomly choosing nodes in every level of the decision tree. Hence, randomization has been proved to be a powerful tool yet again.

Inference:

Randomizing the input for quick sort and the nodes chosen for the decision trees in random forests did help decrease the running time and increase the accuracy than the deterministic algorithm.

Conclusion:

Randomization is not haphazard but is a sequence of random variables that have the limits under a probability distribution but do not produce an output in a deterministic manner. [3] Though it does bring about a change but there is imbalance in prognostic factors because of chance imbalance and with smaller sample sizes, the imbalance will be substantial. Randomization helps testing an algorithm with innumerable test cases and averages out any influence of external variables by evenly distributing them.

Bibliography:

Tables:

Figures:

References:

[1] M.O. Rabin, Probabilistic algorithms, in: J. Traub, ed., Algorithms and Complexity (Academic Press, New York, 1976).

[2] R. Solovay and V. Strassen, A fast Monte-Carlo test for primality, SIAM J. Comput. 6 (1977)

[3] J. Gill, Computational complexity of probabilistic Turing machines, SIAM J. Comput. 6 (1977)

[4] http://courses.washington.edu/inde510/516/AdapRandomSearch4.05.2009.pdf

[5] http://www.immorlica.com/randAlg/Karp91.pdf

[6] http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0132102

[7] http://theory.stanford.edu/people/pragh/amstalk.pdf

[8] http://ieeexplore.ieee.org/document/7355313/

[9] https://github.com/preranakamath/Algorithms_Project