Speeding up Machine Learning using Parallel Processing

Hitesh Kumar
Computer Science Department
San Jose State University
San Jose, CA 95192
408-924-1000
hitesh.kumar@sjsu.edu

ABSTRACT
Serial processing of machine learning algorithms or some its steps takes a lot of time as those steps are running sequentially. Implementing parallel processing helps the machine learning algorithms to get results in a fast and clean manner. Identifying steps that are not dependent on each other and running them parallel on systems or cores is the first step towards parallelization in machine learning. The question arises, which machine-learning algorithm or their pre-processing steps can be parallelized to optimize the overall implementation. This paper identifies and compares different parallelized approaches in machine learning and reviews how much it affects the overall performance.

1. INTRODUCTION
Machine learning is a subset of artificial intelligence that involves the study of the structure of data and its analysis for further modelling processes. Machine learning doesn't follow traditional algorithm approaches, it involves learning from data and based on inputs, predicting outputs through a series of learning. Once the machine learns the data, that machine can take decisions on its own. Many fields have used machine learning as part of their core decision-making processes such as Healthcare, facial recognition in security systems and many more. Machine learning involves a lot of computation as its associated backend algorithm is mathematically inclined and task intensive. Machine learning can be classified into two categories: Supervised and Unsupervised learning.

1.1 Supervised learning
Supervised learning involves learning from data whose label has been already assigned. This type of learning includes comparing the calculated output with actual output and assigning the weights according to the data. For example, labelled data included opcodes of malware files and labels if it's benign or malware [1]. The algorithm learns accordingly and predicts for a valid input is malware or not.

1.2 Unsupervised learning
It involves data whose label is not defined. The algorithm's task is to find the internal structure of the data and solve the required problem. The raw data is input to the algorithm, the algorithms look for any patterns and hidden structures and output the pattern which is interpreted by another machine or a human. This type of learning is very beneficial in recommendation systems and pattern recognition intensive tasks [2].

1.3 Parallel processing
Parallel processing is a technique of segmenting a larger problem into subproblems that are independent in nature and computing their outputs parallelly, using the available computation power. The objective of parallel processing is to increase the throughput and faster execution of overall processes. Parallel processing can be done on various levels Bit-level parallelism is the processing of bits in a parallel fashion, that results in the execution of more instructions and hence reduces the total number of executions in the waitlist. Instruction level parallelism is a hardware-based parallel approach in which the system compiler decides which instruction to run in a static parallel fashion without any deadlocks. Process level parallelism is a computation technique that spawns different processes to multiple cores in the system.

1.4 Core
A core is an essential component of the CPU that performs primary calculations. A CPU can have single, dual, quad or n-cores. More the core, more the performance. The system scheduler performs hyperthreading and achieves parallel processing using the performance of the available cores. Having a multi-core architecture, at a particular time, multiple tasks can execute on multicores, and it increases the throughput [3].

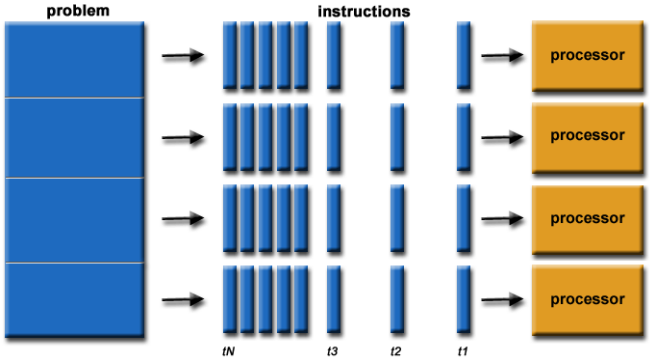

Figure 1. Multiprocessing of instructions to each processor

2. HISTORY
The term ML or machine learning was introduced back in 1959, primarily by Arthur Samuel. A similar word for machine learning was used as self-learning by machines. Also, this field was associated with pattern recognition and pattern classification. There was significant work going on pattern recognition back in the 1970s and 1980s. A breakthrough was the introduction of neural nets which learnt the A-Z alphabets, numbers and some special characters from a computer system.

Currently, the machine learning algorithms serve two purposes- to make classification and to make a prediction. The machines are trained via algorithms to make decisions just like humans and with high computational power. Alan Turing proposed the Turing test which compares the machine and human's decision-making skills. The modern path of machine learning is Artificial Intelligence whose roots are also from the 1980s. After 1980, the recurrent neural network was invented which was a memory-based neural system. in 1985, NetTalk was invented by T.Sejnowski which was a word pronouncing app. Backpropagation came in 1986 which uses reverse mode Automatic Differentiation (AD) to learn the internal structure of data [4]. The efficiency of RNN was improved after the invention of LSTM. There has been a significant improvement and work going on in this field and in upcoming years, there is expected to be big growth in the field of machine learning and artificial intelligence [5].

3. PARALLELIZATION OF ML MODELS

There are multiple machine learning (ML) algorithms that can be parallelized using sklearn python package using n_jobs parameter. Also, we can manually set the n_jobs and cores to check the performance. For performance purposes in this paper, a randomforest classifier is being used with variable features to check the overall performance.

3.1 Parallelization of Random Forest Model

3.*1.1 Random Forest Model*

Random forest is used for classification and regression purposes. It falls under the supervised machine learning category which is easy to implement. A forest consists of collection trees data structures that perform the calculations and build a decision based on the inputs. The best branch of the best tree is selected as the best solution [6]. It has multiple applications ranging from image classification to feature selection.

The algorithm follows below steps:

1. selection of random samples is done
2. Build a decision tree for the sample selected
3. Predict the result for each tree
4. Count the predicted results and get the results with maximum voting.

There are numerous advantages of using the random forest as it is robust and accurate in terms of predictions. It works on both classification and regression problems. This is good for filling missing values [7].

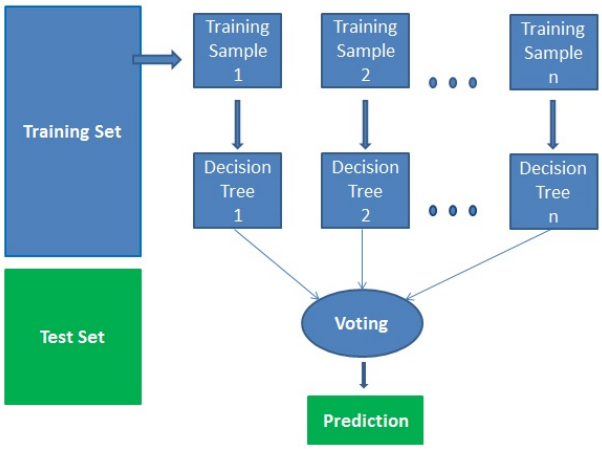

Figure 2. Random Forest classification

3.1.2 Implementation across 8 cores

Here we will implement multicore model training by using a random forest classifier. The number of cores will be changed, and execution time will be recorded for each model training per core. Also, the number of samples are kept fixed which are tabulated below:

Table 1. Model parameters and input size

Model	Records	No of features	Random state
Random Forest	100K	30	5

```
No of cores=1: took 210.384 seconds
No of cores=2: took 107.274 seconds
No of cores=3: took 74.202 seconds
No of cores=4: took 57.760 seconds
No of cores=5: took 53.273 seconds
No of cores=6: took 48.729 seconds
No of cores=7: took 45.470 seconds
No of cores=8: took 42.705 seconds
```

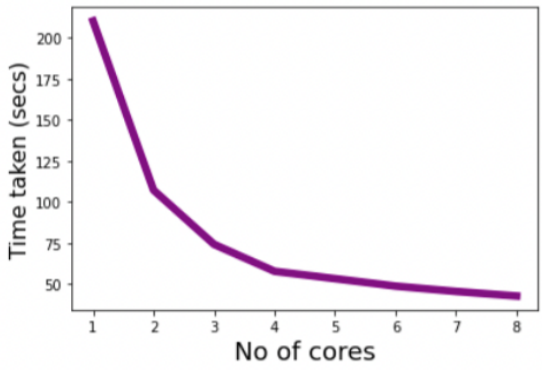

Figure 3. Graph showing relationship between core usage and time taken to execute the model for 100,000 records for random forest model

This time, the records are reduced to only 100 with 30 features and then performance is evaluated. The cores are kept total 8 and the performance is calculated using the same python code:

Table 2. Model parameters and input size

Model	Records	No of features	Random state
Random Forest	100	30	3

```
No of cores=1: took 0.369 seconds
No of cores=2: took 0.372 seconds
No of cores=3: took 0.367 seconds
No of cores=4: took 0.380 seconds
No of cores=5: took 0.388 seconds
No of cores=6: took 0.394 seconds
No of cores=7: took 0.394 seconds
No of cores=8: took 0.401 seconds
```

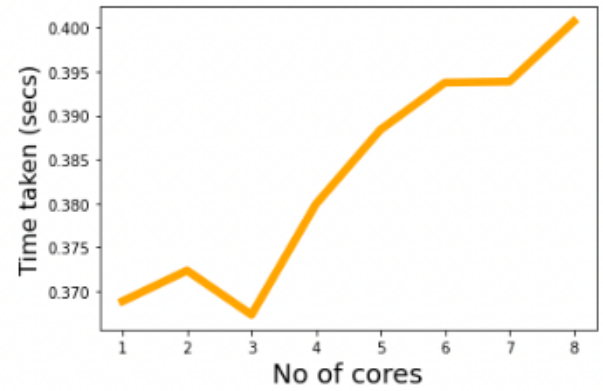

Figure 4. Graph showing relationship between core usage and time taken to execute the model for only 100 records for random forest model

Below python code is used for implementing the relationship between number of cores and execution speed for random forest model training:

```
#number of cores vs execution speed
#author : Hitesh Kumar
#system : MAC M1
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from matplotlib import pyplot
from time import time

# dataset
X, y = make_classification(n_samples=100000, n_features=30, n_informative=15, \
                           n_redundant=5, random_state=2)
results = list()

cores = [1, 2, 3, 4, 5, 6, 7, 8] #number of cores
for n in cores:
    start = time() # start time
    model = RandomForestClassifier(n_estimators=500, n_jobs=n)
    model.fit(X, y) # fit the model
    end = time()
    result = end - start # end time
    print('No of cores=%d: took %.3f seconds' % (n, result))
    results.append(result)
pyplot.plot(cores, results,linewidth = '5.5',color='purple')

pyplot.xlabel('No of cores', fontsize=18)
pyplot.ylabel('Time taken (secs)', fontsize=16)
pyplot.show()
```

Figure 5. Python implementation of Random Forest for 100,000 records

3.2 Parallelization of K-Fold cross validation

3.2.1 K-Fold Cross Validation (CV)

K-Fold is a gold standard resampling procedure for model evaluation when the data is less or there are limited rows. The procedure is easy to implement which splits into K groups and select one for training and K-1 groups for training and keeps iterating for other groups for train and test. Each time, a score is calculated, and the model performance is evaluated. The advantage of using this method is that it removes any bias from the data and created a perfect methodology for test and train split [8] [9]. The below is the procedure:

Step 1: Randomly shuffle the dataset with a seed value

Step 2: Split the dataset into k subgroups

Step 3: Select the first group for testing and the rest for training

Step 4: Keep iterating all the groups by selecting one for testing and the rest for training

Step 5: Calculate the score each time

The value of 'k' is usually set depending on the dataset size. Usually, the value of k is set 10 which is experimentally found stable for bias removal. As the value of k increases, there are more splits and the group's size get smaller and hence the bias reduces [10] [11].

Let us assume we have a match set and nonmatch set of two samples. The splitting follows below:

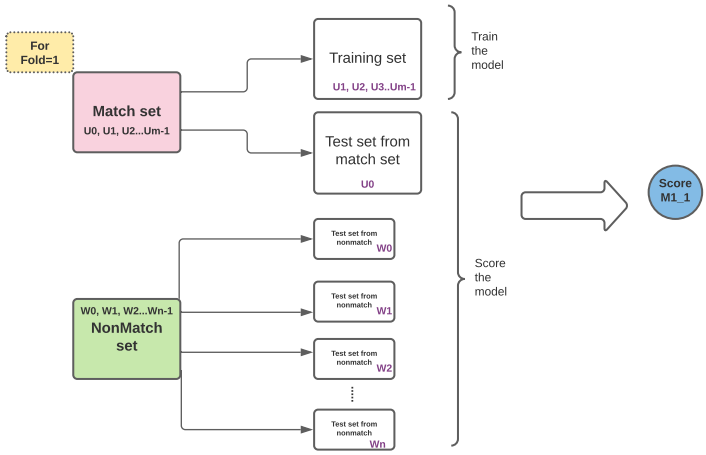

Figure 6. Cross Validation of dataset

3.2.2 Implementation of CV across 8 cores

For the implementation of K-fold CV, n_jobs parameter is used for python sklearn package. The initial value is set 1 and different values are set from 1-8 to check the overall throughput of the program. The sklearn package gives the flexibility to work on different cores. The function cross_val_scores() is initiated from sklearn.model_selection package. For each core, CV is done across different cores i.e., the model is trained across different cores which increases the throughput. The time is calculated for each increase of core execution. The parameters and model is below tabulated with input information and dataset size:

Table 3. Model parameters and input size

Cross Validation	Records	No of features	Random state
K-Fold, K=10	10,000	30	5

```
No of cores=1: took 77.190 seconds
No of cores=2: took 40.274 seconds
No of cores=3: took 27.689 seconds
No of cores=4: took 21.712 seconds
No of cores=5: took 19.504 seconds
No of cores=6: took 17.681 seconds
No of cores=7: took 17.002 seconds
No of cores=8: took 15.140 seconds
```

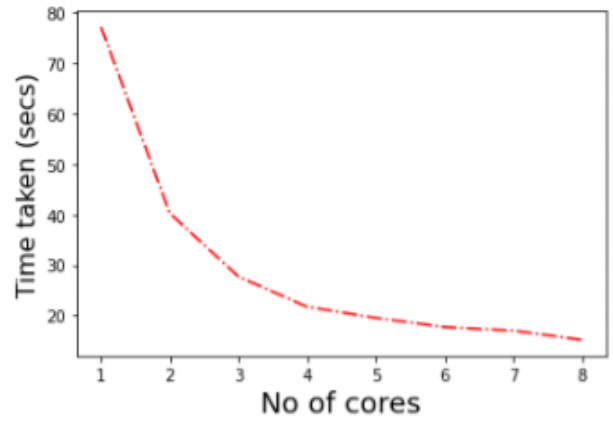

Figure 7. Time vs Cores for Cross validation of K=10 and 10,000 records

To test the same analogy as of random forest, 100 records were taken, and the performance graph is plotted. All the parameters are kept same except total number of records.

Table 4. Model parameters and input size

Cross Validation	Records	No of features	Random state
K-Fold, K=10	10	30	5

```
No of cores=1: took 2.124 seconds
No of cores=2: took 2.222 seconds
No of cores=3: took 1.451 seconds
No of cores=4: took 0.773 seconds
No of cores=5: took 1.381 seconds
No of cores=6: took 0.957 seconds
No of cores=7: took 0.958 seconds
No of cores=8: took 0.878 seconds
```

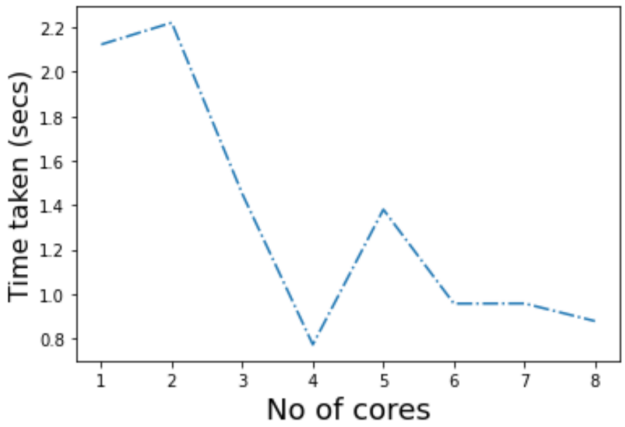

Figure 8. Time vs Cores for Cross validation of K=10 and 100 records

Below python code is used for implementing the relationship between number of cores and execution speed for K-Fold cross validation:

```
# Cross Validation : number of cores vs execution speed
#author : Hitesh Kumar
#system : MAC M1
from time import time
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from matplotlib import pyplot
# dataset
X, y = make_classification(n_samples=10000, n_features=30, n_informative=15,\
                           n_redundant=5, random_state=5)
results = list()
# number of cores
cores = [1, 2, 3, 4, 5, 6, 7, 8]
for n in cores:
    model = RandomForestClassifier(n_estimators=100, n_jobs=1)
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    start = time()
    n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=n)
    # record the current time
    end = time()
    result = end - start
    print('No of cores=%d: took %.3f seconds' % (n, result))
    results.append(result)
pyplot.plot(cores, results,'-.',color='red')
pyplot.xlabel('No of cores', fontsize=18)
pyplot.ylabel('Time taken (secs)', fontsize=16)
pyplot.show()
```

Figure 9. Python implementation of cross validation for 10,000 records

4. CONCLUSION

For model training using 8-cores for Random Forest, there is a significant decrease in time for 100K thousand records with 30 features. It is easily visible that there is a steady fall in execution time which falls sharply up to 4 physical cores and gets a steady fall after the 5th core. The core-1 took 210.384 seconds, core-4 took 57.760 seconds and core-8 took 42.705 seconds.

It is evident that till 4 core usages, the performance is very good but after 4 cores, there is not much performance gain. On the other hand, if the records are merely 100, the performance graph is the opposite. It seems that with lower records, the parallel approach doesn't work at all as the management across multiple cores is itself another task. Hence the total time increases with every addition of cores in random forest model training.

In the case of cross-validation, for 10K records, the total time taken for completion is 77 seconds with core-1 and using all 8 cores the time taken is only 15 seconds. On the other hand, when the records are reduced, it is hard to deduce as the graph does not follow any upward or downward trend.

Therefore, from the above observations, it is evident that if the number of records is in the thousands, the performance gain is substantial as compared to a few records where multiprocessing is not much helpful.

5. REFERENCES

[1] A. Ahmed, I. Zualkernan, and H. Elghazaly, "Unsupervised Clustering of Skills for an Online Learning Platform," presented at the - 2021 International Conference on Advanced Learning Technologies (ICALT), 2021, pp. 200–202, doi: 10.1109/ICALT52272.2021.00066.

[2] L. von Rueden *et al.*, "Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems." p. 1, 2021.

[3] E. Ramalakshmi and N. Kompala, "Multi-threading image processing in single-core and multi-core CPU using R language," presented at the - 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), 2017, pp. 1–5, doi: 10.1109/ICECCT.2017.8117873.

[4] A. Plasek, "On the Cruelty of Really Writing a History of Machine Learning," vol. 38, no. 4. pp. 6–8, 2016.

[5] A. Shukla, L. Manoael, and T. Wang, "Hybrid and Ensemble-Based Personalized Recommender System - Solving Data Sparsity Problem," presented at the - 2021 Third International Conference on Transdisciplinary AI

[6] Z. Bingzhen, Q. Xiaoming, Y. Hemeng, and Z. Zhubo, "A Random Forest Classification Model for Transmission Line Image Processing," presented at the - 2020 15th International Conference on Computer Science & Education (ICCSE), 2020, pp. 613–617, doi: 10.1109/ICCSE49874.2020.9201900.

[7] Y. Zhang and B. Luo, "Parallel classifiers ensemble with hierarchical machine learning for imbalanced classes," presented at the - 2008 International Conference on Machine Learning and Cybernetics, 2008, vol. 1, pp. 94–99, doi: 10.1109/ICMLC.2008.4620385.

[8] T. Shinozaki and M. Ostendorf, "Cross-Validation EM Training for Robust Parameter Estimation," presented at the - 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07, 2007, vol. 4, pp. IV-437-IV–440, doi: 10.1109/ICASSP.2007.366943.

[9] Y. Zhao and K. Wang, "Fast cross validation for regularized extreme learning machine," vol. 25, no. 5. pp. 895–900, 2014.

[10] L. Lingqiao, Y. Huihua, H. Qian, Z. Jianbin, and G. Tuo, "Design and Realization of the Parallel Computing Framework of Cross-Validation," presented at the - 2012 International Conference on Industrial Control and Electronics Engineering, 2012, pp. 1957–1960, doi: 10.1109/ICICEE.2012.520.

[11] Jamil-Ur-Rehman and Y. Zhang, "Fast intra prediction mode decision using parallel processing," presented at the - 2005 International Conference on Machine Learning and Cybernetics, 2005, vol. 8, pp. 5094-5098 Vol. 8, doi: 10.1109/ICMLC.2005.1527841.