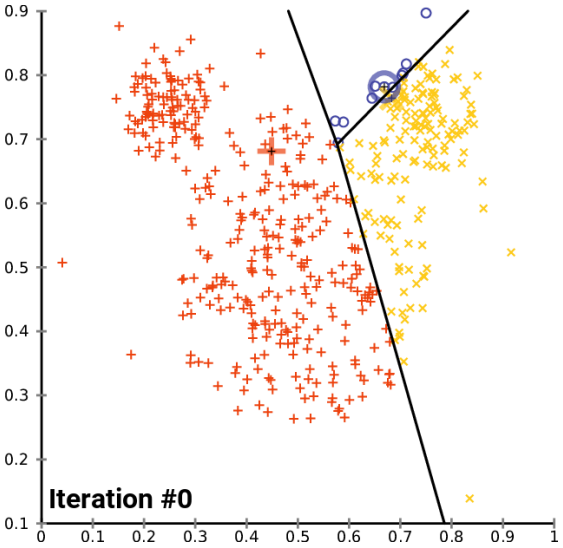
OpenMP vs MPI

OpenMP mainly used on single core(multithreading) whereas MPI uses all cores of CPU. OpenMP didn't yield positive result for k-means clustering and merge sort hence shift to MPI was made.

K-Means



***k*-means clustering** is a method that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. *k*-means clustering minimizes within-cluster variances.

The problem is computationally difficult (NP-hard); however, efficient heuristic algorithms converge quickly to a local optimum. *k*-means clustering tends to find clusters of comparable spatial extent.

Naïve K Means Algorithm- Given an initial set of *k* means $m_1^{(1)},\ldots,m_k^{(1)}$, the algorithm proceeds by alternating between two steps:

- **Assignment step**: Assign each observation to the cluster with the nearest mean, that is with the least squared Euclidean distance.Each observation is assigned to exactly one cluster center even if it could be assigned to two or more of them. O (N * K) distance computations are needed to find the closest cluster center.
- **Update step**: Recalculate means (centroids) for observations assigned to each cluster. O(N) computations are needed to determine the new centroids.

The algorithm has converged when the assignments no longer change. The algorithm is not guaranteed to find the optimum. The algorithm is often presented as assigning objects to the nearest cluster by distance.

Parallelizing K Means Algorithm

The most time-consuming step in the K Means algorithm is the Assignment step. And each data point is independent in its computation of nearest cluster center. To parallelize this step, data points are divided between different processes and each process computes the nearest cluster center for the points assigned to the process.

Update step requires all the memberships assigned to be used and that's where aggregation of information from different processes takes place.

Time Comparisons

For 4 clusters

Parallel :

Computation time = 0.000858 sec

Computation time = 0.000855 sec

Computation time = 0.000853 sec

Computation time = 0.000853 sec

Sequential :

Computation time = 0.001769 sec

Sequential Time/Parallel Time = 0.001769 / 0.000858 = 2.06177156177

For 6 clusters

Parallel :

Computation time = 0.001481 sec

Computation time = 0.001450 sec

Computation time = 0.001454 sec

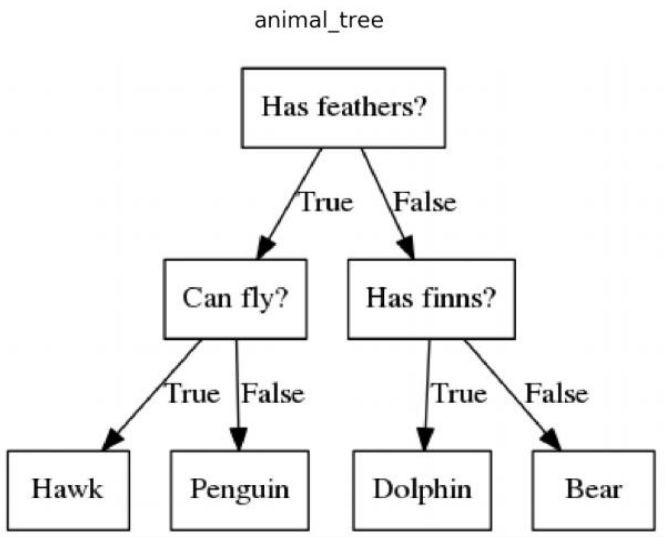
Computation time = 0.001458 sec

Sequential:

Computation time = 0.004387 sec

Sequential Time/Parallel Time = 0.004387 / 0.001481 = 2.96218771101

Decision Tree



The goal is to create a model that predicts the value of a target variable based on several input variables. A decision tree is a simple representation for classifying examples. We assume that all of the input features have finite discrete values, and there is a single target feature called the "class".

A decision tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the class or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class signifying that the data set has been classified by the tree into a specific class.

A tree is built by splitting the source set, constituting the root node of the tree, into subsets—which constitute the successor children. Information Gain, which is a measure of impurity, is computed for each attribute (and its attribute values). The attribute which results in the highest information gain is selected.

This process is repeated on each derived subset in a recursive manner. The recursion is completed when the subset at a node has all the same values of the class, or when splitting no longer adds value to the predictions (for example, #data points in the subset are too small). This process of *top-down induction of decision trees* (TDIDT) is an example of a greedy algorithm, and it is by far the most common strategy for learning decision trees from data.

Parallelization - The most time-consuming computation in decision tree learning is in the selection of the attribute to split the data set on. This process of selecting an attribute is parallelized by enabling each process to separately compute the Information Gain for a specific attribute. After information gain values are available for each attribute, the attribute with the maximum information gain is selected.

Here the number of processes should be more than the number of attributes in the dataset. Of course, more than 1 attribute can be assigned to the process, that has not implemented in the current version.

Notes on MPI
Number of processors as command line input should exceed the number of attributes
For determining an attribute to split on, unique processor is assigned for each attribute
And all the information gain values are collated to determine the attribute to split on.
Observation : On a dual core computer time taken came out to be higher than a simple decision tree
as for the synthetic dataset number of attributes is 6. Performance improved on 8-core machine.

Datasets
1. 1727 data points and 6 different attributes, 4 class values
2. 30 data points and 3 different attributes, 2 class values

Output and Time Comparisons (for dataset1(large))

Simple
time:0.021024
in testing
Correct 194 Incorrect 13 Unexpected 0

Parallel
rank 0 in output time computation 0.008744

rank 1 in output time computation 0.008922
rank 2 in output time computation 0.008928
rank 3 in output time computation 0.009007
rank 4 in output time computation 0.008988
rank 5 in output time computation 0.008847
rank 6 in output time computation 0.009034
Correct 194 Incorrect 13 Unexpected 0
Correct 194 Incorrect 13 Unexpected 0
Correct 194 Incorrect 13 Unexpected 0
Correct 194 Incorrect 13 Unexpected 0
Correct 194 Incorrect 13 Unexpected 0
Correct 194 Incorrect 13 Unexpected 0
Correct 194 Incorrect 13 Unexpected 0

Time comparison = Non-parallel Time / Parallel Time(maximum among all ranks) =
0.021024/0.009034 = 2.32720832411

Output and Time Comparisons (for dataset2(small))

Simple:
time:0.001102
in testing
Correct 30 Incorrect 0 Unexpected 0

Parallel:

rank 0 in output time computation 0.000159
rank 1 in output time computation 0.000162
rank 2 in output time computation 0.000162
rank 3 in output time computation 0.000162
Correct 30 Incorrect 0 Unexpected 0
Correct 30 Incorrect 0 Unexpected 0
Correct 30 Incorrect 0 Unexpected 0
Correct 30 Incorrect 0 Unexpected 0

Time comparison = Non-parallel Time / Parallel Time(maximum among all ranks) =
0.001102/0.000162 = 6.8024691358

Output and Time Comparisons (for Standard Titanic Dataset)

Simple:
time:0.037771
in testing
Correct 66 Incorrect 34 Unexpected 7

Parallel:

rank 0 in output time computation 0.004969
rank 1 in output time computation 0.004944

rank 2 in output time computation 0.004957
rank 3 in output time computation 0.004936
rank 4 in output time computation 0.004943
rank 5 in output time computation 0.004957
rank 6 in output time computation 0.004952
rank 7 in output time computation 0.004962
Correct 66 Incorrect 34 Unexpected 7
Correct 66 Incorrect 34 Unexpected 7
Correct 66 Incorrect 34 Unexpected 7
Correct 66 Incorrect 34 Unexpected 7
Correct 66 Incorrect 34 Unexpected 7
Correct 66 Incorrect 34 Unexpected 7
Correct 66 Incorrect 34 Unexpected 7
Correct 66 Incorrect 34 Unexpected 7

Time comparison = Non-parallel Time / Parallel Time(maximum among all ranks) = 0.037771/0.004969 = 7.60132823506

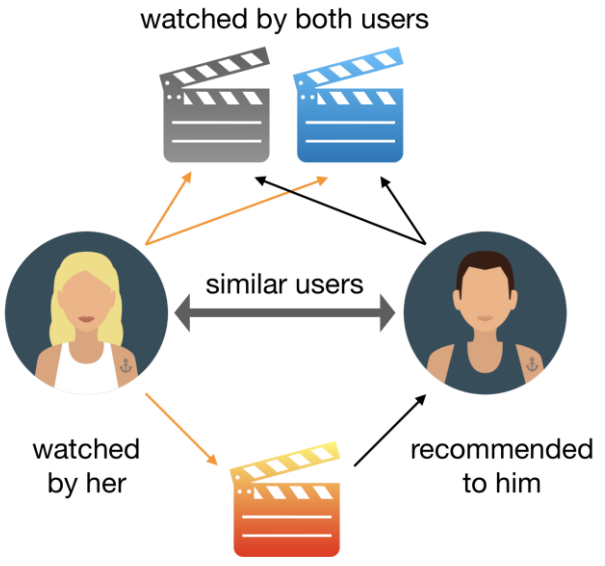
Information Gain for Decision Trees

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

Item-Item Collaborative Filtering



Item-item collaborative filtering for recommender systems is based on the similarity between items calculated using people's ratings of those items. It was invented and used by Amazon.com in 1998. Item-item models use rating distributions per item, not per user. Let's consider a collection of N users and a collection of M items. User ratings can be represented by a N × M matrix. An item is represented by the ratings assigned to the item by all the users (N * 1 dimensional vector). With more users than items, each item tends to have more ratings than each user, so an item's average rating usually doesn't change quickly.

1. The system executes a model-building stage by finding the similarity between all pairs of items. This similarity function can take many forms, such as correlation between ratings or cosine of those rating vectors. We compute the movie similarity Matrix using the standard MovieLens 1M Data Set. To compute similarity between two movies, X and Y, we compute Cosine Similarity between these two vectors.

Movie similarity matrix is M * M matrix. Computing this matrix is $O(M^2 * N)$. However, computation of this matrix is highly parallelizable – value in each cell of the matrix depends on two user vectors and nothing else. We use a simple logic to allocate the computation to each of the processes.

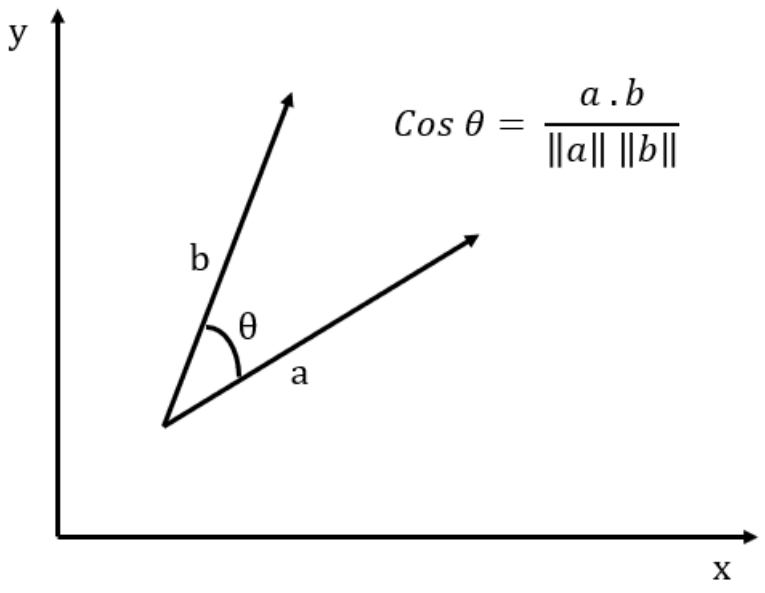
2. The system executes a recommendation stage. It uses the most similar items to a user's already-rated items to generate a list of recommendations. Usually this calculation is a weighted sum. This form of recommendation is analogous to "people who rate item X highly, like you, also tend to rate item Y highly, and you haven't rated item Y yet, so you should try it".

The score to an unseen movie is computed based on the similarity of the movies and the previous ratings provided by the target user. If $S_{i,j}$ is the similarity between movie i and movie j, $V_{u,j}$ is the rating given by user U to movie j. The predicted rating for movie j for user U is given by

$$v_{ui}^{*} = \frac{\sum_{j} s_{ij} * v_{uj}}{\sum_{j} s_{ij}}$$

Scores assigned to unseen movies are sorted and then combined to create a recommendation list.

	Movies					
Users	Forrest Gump	Cast Away	Captain Philips	The Terminal	The Terminator	The Matrix
A	1	1	1	1	0	0
B	1	1	1	?	0	0
C	0	0	0	?	1	1



$$Cos\ \theta = \frac{a\ .\ b}{\|a\|\ \|b\|}$$

OUTPUT of the code(takes long time to run due to large size of the dataset)

Parallel:

number of movies read 3883

number of movies read 3883

number of movies read 3883

number of movies read 3883

number of movies read 3883

number of movies read 3883

number of movies read 3883

number of movies read 3883

number of ratings read 1000209 number of Users 6040

number of ratings read 1000209 number of Users 6040

number of ratings read 1000209 number of Users 6040

number of ratings read 1000209 number of Users 6040

number of ratings read 1000209 number of Users 6040

number of ratings read 1000209 number of Users 6040

number of ratings read 1000209 number of Users 6040

number of ratings read 1000209 number of Users 6040

number of processors 8

rank 3 count 942356

rank 4 count 941874

rank 1 count 964316

rank 2 count 941871

rank 6 count 941876

rank 7 count 942362

rank 5 count 942360

rank 0 count 1018517038

rank 5 in output time computation 72.258850

rank 7 in output time computation 72.269423

rank 0 in output time computation 72.261055

rank 6 in output time computation 72.275681

rank 1 in output time computation 72.284463

rank 2 in output time computation 72.285918

rank 3 in output time computation 72.288311

rank 4 in output time computation 72.272928

Recommendation for user 3245

Rank | Movie Name

1 | Toy Story (1995)

2 | Contender, The (2000)

3 | Two Family House (2000)

4 | Tigerland (2000)

5 | Requiem for a Dream (2000)

Simple:

number of movies read 3883

number of ratings read 1000209 number of Users 6040

number of processors 1

rank 0 count 7536903

rank 0 in output time computation 192.892791

Recommendation for user 3245

Rank | Movie Name

1 | Toy Story (1995)

2 | Contender, The (2000)

3 | Two Family House (2000)

4 | Tigerland (2000)

5 | Requiem for a Dream (2000)

Time comparison = 192.892791/72.288311 = 2.66838148978