MACHINE LEARNING BASED 2048 SOLVER

A comparison of various models

Team Members

Rahul Aravind Mehalingam(rxm151730)

Praveen Erode Murugesan (pem150030)

Abstract:

The aim of the project is to develop a machine learning model to solve 2048 game. The main challenge of this problem is to find the sequence of optimal states that leads to victory in the game. Most of the existing machine learning based solver makes use of Q learning and faces the problems such as huge state space and choice of optimal reward function. In order to overcome, this project proposes a Markov Random Field model to solve the 2048 game board. In addition various other models are also implemented and their performances are studied.

About the Game:

2048 is played on a gray 4×4 grid, with numbered tiles that slide smoothly when a player moves them in four different directions. Every turn, a new tile will randomly appear in an empty spot on the board with a value of either 2 or 4. The game is designed in such a way that the probability of occurrence of 2 is more than that of 4. Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the border of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that conjoined. The resulting tile cannot merge with another tile again in the same move unless they both have the same values[1].

Existing Work:

There are a couple of existing works that were carried out on solving the 2048 game.

i) Q-Learning:

Q-learning is a part of a family of reinforcement techniques known as temporal difference learning. They try to learn a value function by bootstrapping and iteratively improving upon an initial guess. Essentially, a value function takes a state and action pair and gives an idea of how much reward that can be accumulated if you perform that action in that state and follow the strategy of choosing the actions that gives the maximum reward value. This concept can be applied to solve this game where in the board states correspond to the 4x4 grid tile state, the actions can be any one of the directions such as top, down, left and right, and the reward value can be the score of the board state after performing an action[2].

Disadvantages:

The state space is unbelievably huge. It's about 12^{16}. There are a total of 16 tiles. Each tile can take upto 12 values. The domain of the values are {2^0, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048} with 2^0 being the empty tile. With such a large state space, storing the value function as a table for each board state is a not so feasible technique. Then Q-learning is not

interested in maximizing single step rewards but rather getting the most reward over time. The reinforcement learning would probably work but it requires a good deal of thought and effort to decide the reward function[2].

ii) Q-Learning with Neural Networks as function approximator:

Neural networks are used as some type of function approximator of reward function. State and action representation are very important for neural networks and exploiting the symmetry in the game would make it much a simpler problem. The neural network would take a state, s(t), and an action, a(t), as input and give an output the value of that state action pair, Q(s(t),a(t)). The neural network is trained with all possible values of action for a particular state and then an appropriate action for which the value is maximum, is picked. Therefore, a new state and a reward associated with it, is obtained. The update rule for Q-learning is that the value of Q(s(t),a(t)) is updated to a little closer to the value r(t+1) + D*max{Q(s(t+1),a(t+1)}, where 0<D<=1 is a discount factor and the max is taken over all possible a(t+1). This update can be performed on the neural network by using T as a target value in a loss function, such as $(Q(s(t),a(t)) - T)^2$ by using squared error, computing the gradients via backpropagation, and changing the weights with a step in the opposite direction of the gradient. As for directly working with the reward, reinforcement learning is not interested in maximizing single-step rewards but rather getting the most reward over time. So instead of approximating the value function, the total reward is maximized by approximating a gradient of the reward with respect to the network weights via finite differences. Hence the neural network plays a bunch of games and average the return. Then accordingly, the changes are made to the network weights and another bunch of games are played again. The derivative with respect to the weight is then approximated by dividing the difference in the two returns divided by the change in weight[2].

Disadvantages:

Training would take much longer time. With the current representation, the game has to be played several million times and an intensive training is required. The network would take much longer time to converge to the value function. Additionally while the network is training, the weight changes need to be accumulated and has to be applied. Unless you somehow control the states that the network sees, states encountered later on in the game will likely not get visited often or ever, leading to reduced performance[2].

AI Solver:

2048 game is solved using AI solver. The game is discrete state space, turn based game like chess or checkers. As same as that of Chess, the solver makes use of Min-Max search and alpha-beta pruning. The algorithm makes use of three main heuristics. They are[3]

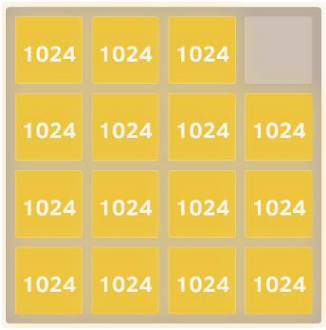
Monotonicity:

The values of the tiles are either increasing or decreasing along the right/left or up/down directions. This ensures that the high valued tiles are always at the corner and prevents the small values tiles from getting stuck in the middle or corners of the board without being able to get merged.

As you can see the above picture, In the 3rd column, the smallest number tile is getting stuck in between 128 and 16. Such situations can be avoided in AI solver using the monotonicity heuristic.

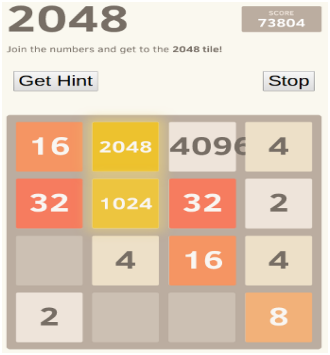
Smoothness:

In order to merge tiles, their values must be equal. This heuristic maximizes the count of equal neighboring tiles.



Free Tiles:

The board must always have as many free tiles as possible so as to accommodate more new numbers and provide way for those numbers to get merged and form larger numbers.



Tools Used:

AI solver that was designed by Gabriele Cirulli made use of various heuristics to solve the 2048 game. The heuristics are as follows,

- The board should maintain the property of monotonicity (Monotonicity)
- If there are same numbers that are adjacent, it needs to be merged (Smoothness)
- At any state, board must have as many empty tiles as possible (Free tiles).

Programming Language: Java, Octave

Packages: Selenium

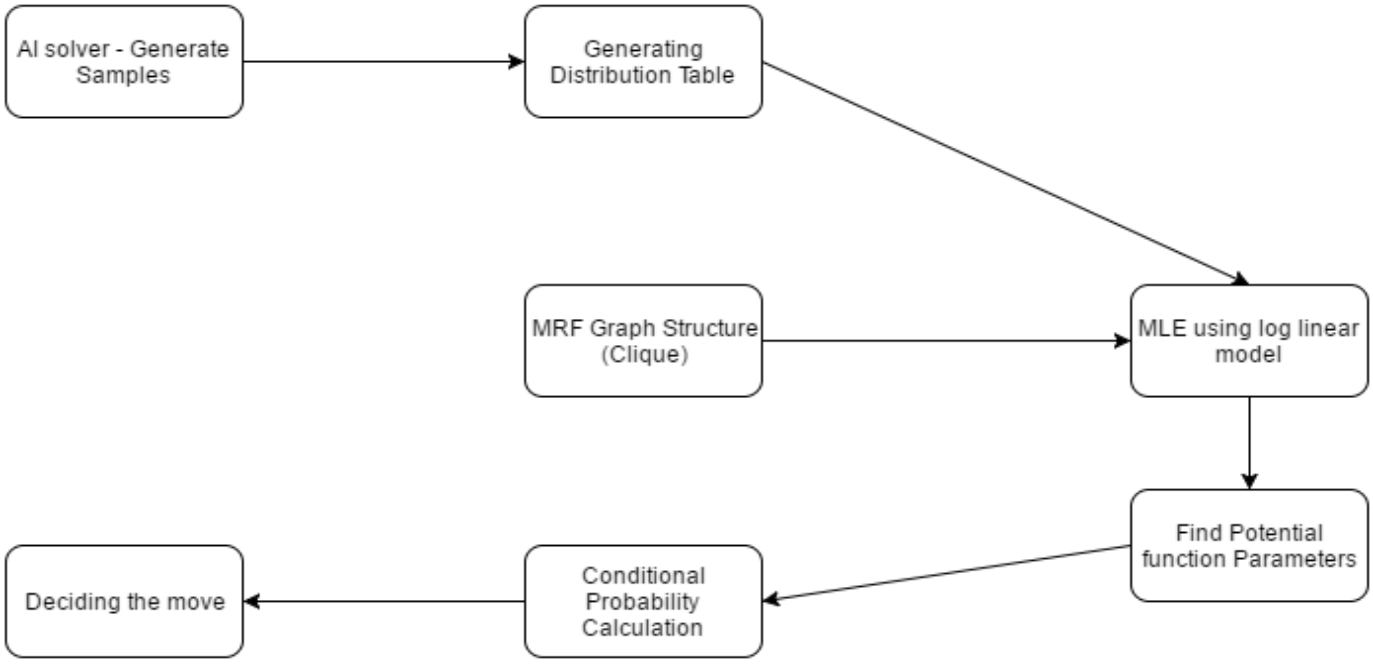
Data set generation:

The 2048 AI solver is used to generate samples. This solver provides samples that are well distributed. It provided various kinds of samples as it had different levels of play. We can choose levels such as dumb, random, smart and so. These options generated a well distributed data set. As these data were to be read from a website, it necessitated the use of a java package called Selenium. Selenium is useful in manipulating the HTML element. Thus the different board states are captured and written into a file.

Proposed Solution:

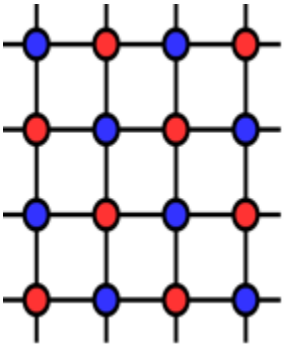
The idea is to apply Markov Random Fields model to design the game. The square lattice model is used is to represent the board.

Work-Flow diagram:



Modelling:

The 2048 game board is modelled as Square Lattice in which each node represents a tile in the game board. It is a graphical model that represents each node that is dependent only on its neighbors. This model is chosen because, the value of each tile depends only on its adjacent ones. The value in a tile gets changed only if there are same numbers in any one of the adjacent tiles and gets merged. Thus the square lattice model depicts the game board perfectly. In addition to the adjacent neighbors each tile is also associated with the final outcome. It is depicted as follows.



The training samples are represented with this model. The usage of Markov Random field necessitates the learning of the potential function. The nodes in the square lattice model along with the classification label are connected to form different cliques. There are 24 edges in the board connecting different nodes. And each node is connected to the classification label y. Thus 24 cliques each of size three is formed. Each clique has a potential function associated with it.

$$p(x|\theta) = \frac{1}{Z(\theta)} \prod_C \psi_C(x_C|\theta)$$

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_{c \in C} \psi_c(x_c)$$

$$Z = \sum_{x'_1,\dots,x'_n} \prod_{c \in C} \psi_c(x'_c)$$

The potential functions are represented using one of the log linear family of functions as follows for computational easiness.

$$\log l(\theta) = \left[\sum_m \sum_C \log \psi_C(x_C^m|\theta) \right] - M \log Z\,(\theta)$$

$$p(x|\theta) = \frac{1}{Z} \prod_C \exp(\theta_C(x_C))$$

Thus, it has reduced the problem of learning the board states to learning the parameters of the potential function. Once the potential functions' parameters are learned, the probability of the board state could be found for a particular classification label.

$$p(Y \mid X) = \frac{1}{Z(x)} \prod_{c \in C} \psi_c(x_c, y_c)$$

$$Z(x) = \sum_{y'} \prod_{c \in C} \psi_c(x_c, y'_c)$$

For any particular board state , unless the game is completed, there exists four possible moves in the up, down, right and left directions are possible. The board states of all those movements are found and the corresponding probability of winning the game from that board states are computed. Then the board state that has highest winning probability is chosen and the corresponding move is made. This continues till the game is completed.

Implementation:

In implementing the system, many experimental modelling is done in addition to the proposed Markov random fields to compare the efficiency of different models.

Experimented Models:

Naive Bayes Model:

Each tile of the game board is represented as a feature t_i and the classification label y represents a classification (win or loss).

$$\mathbf{P(Y=win/t_1....t_{16}) = P(Y = win) *P(t_1/Y)*..... P(t_{16}/Y)}$$

The probability of win from a particular board state is given by the above formula. The probability of all the four possible states are computed and the state with the maximum probability is chosen and the corresponding move is made in the game board.

Since the Naïve Bayes model did not consider the values in the adjacent tiles, this model failed to capture the necessary probabilities to win the game.

Bayesian Network Chow Liu trees:

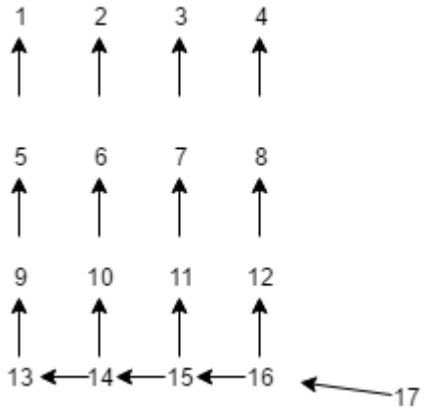
The Bayesian network Chow Liu tree is used for structure learning. Similar to the above method, this approach also had sixteen nodes to depict the tiles and a node for the classification label. A complete graph is constructed with these seventeen nodes. Thus each node is connected to all the other node with sixteen edges. The mutual information between the nodes is assigned as the edge weight. A maximum spanning tree is formed from the graph. As there is no dedicated algorithm to find the maximum spanning tree, the weights of the edges are negated and the minimum spanning tree is found Edmonds Branching Algorithm. The tree has the classification label as its root. This tree give rises to a directed Bayesian network. The winning probability of the board state is computed using this tree.

$$\mathbf{P(Y = win / Maximum\ Spanning\ tree) = P(Y = win) * \pi\ P(t_i / Parent(t_i))}$$

Similar to the above approach the probabilities of the four next possible states are found and the best one is chosen and the corresponding move is made.

Another Bayesian Network:

The following Bayesian model was designed and the samples were



The conditional probabilities for the nodes in the Bayesian network are learnt using the samples and the game is played.

Markov Random Field:

In this approach the board is represented in a square lattice model. The classification label is connected to each of the sixteen nodes. According to this model, each node will have 12 parameters associated with it for each of the 12 values it can take. All edges except the ones from the classification label node connects two (nodes) representing tiles. Thus each edge can have 12*12 possible states. Each such state has a parameter associated with it. The edges connecting classification label will have 2*12 possible states and equal number of parameters associated with it. The joint probability is calculated as

$$P(x_1,x_2 \ldots x_{16},y) = e^{w_y{}^y}\, \pi\, e^{w_i{}^{x_i}}\, \pi^{w_{i,j}{}^{x_i x_j}}$$

The conditional probability is computed as follows

$$P(Y=win/\ x_1,x_2,\ldots x_{16}) = P(x_1,x_2,\ldots x_{16} / Y=win) / \sum P(x_1,x_2,\ldots x_{16} / Y)$$

$$P(Y=win/\ x_1,x_2,\ldots x_{16}) = e^{w_y{}^y}\, \pi^{w_{i,y}{}^{x_i y}} \ /\ \sum e^{w_y{}^y}\, \pi^{w_{i,y}{}^{x_i y}}$$

It is observed that this model finally gets reduced almost similar to that of Naïve Bayes and hence this method also has not produced the expected result.

Note: Node17 in diagrams represents the classification label

Win or Loss classification based:

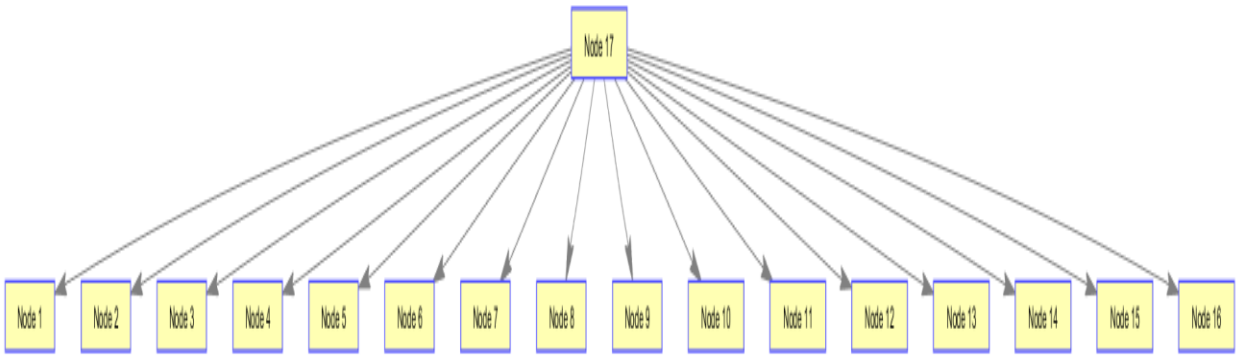
The four actions are performed on the current state and the state which produces the maximum probability is considered and the same strategy is repeated.

Move based classification:

The samples generated by the AI solver are labelled with the directions. The conditional probabilities are learnt with Bayesian networks and the game is played. With the given probabilities, the board state is labelled with a particular action and the corresponding action is executed.

Experimental Results:

Naïve Bayes model:



Accuracy:

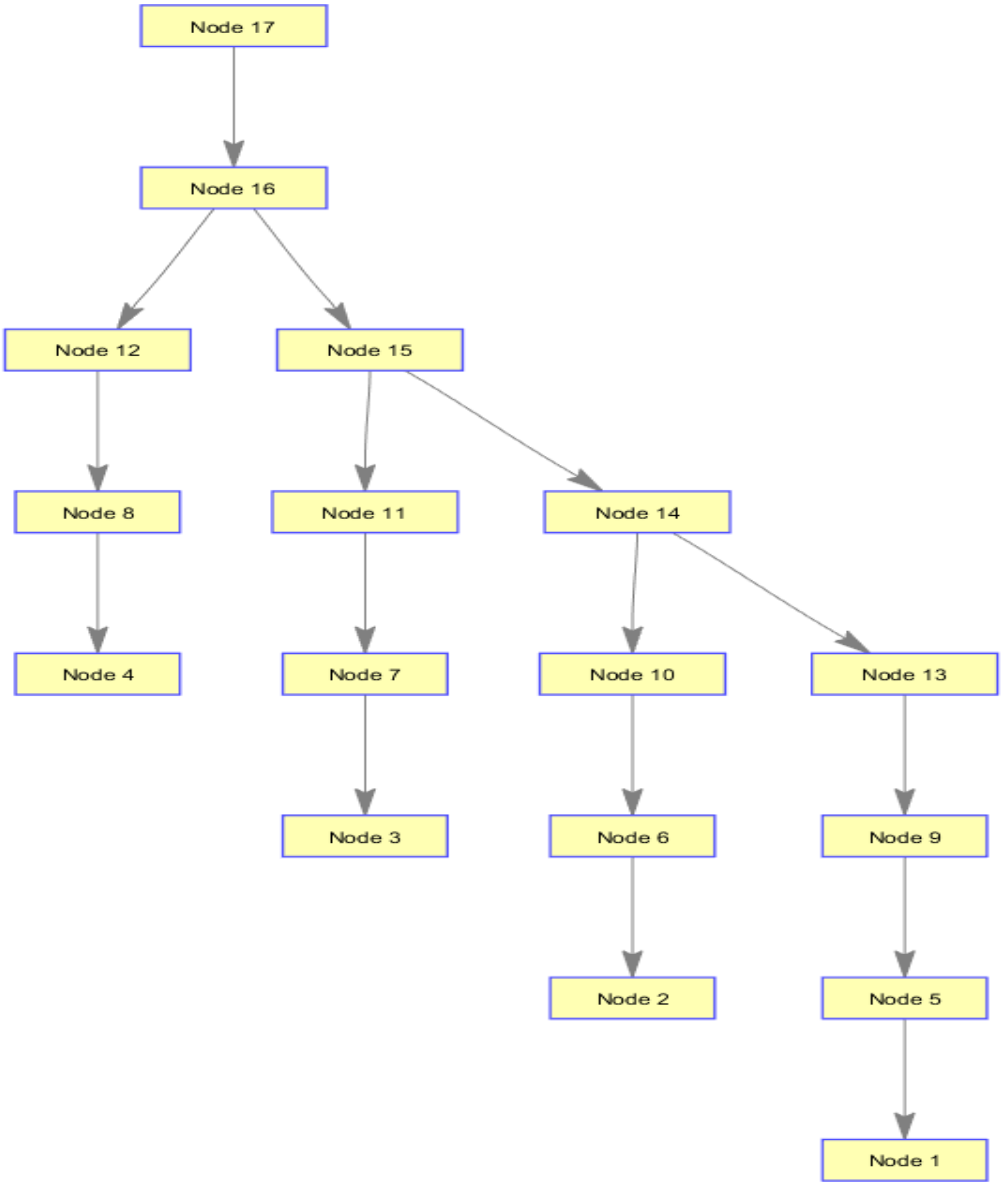
Move based classification accuracy: 38.36 %

Win or loss based classification accuracy: 63.53%

Bayesian Network Chow Liu Trees:

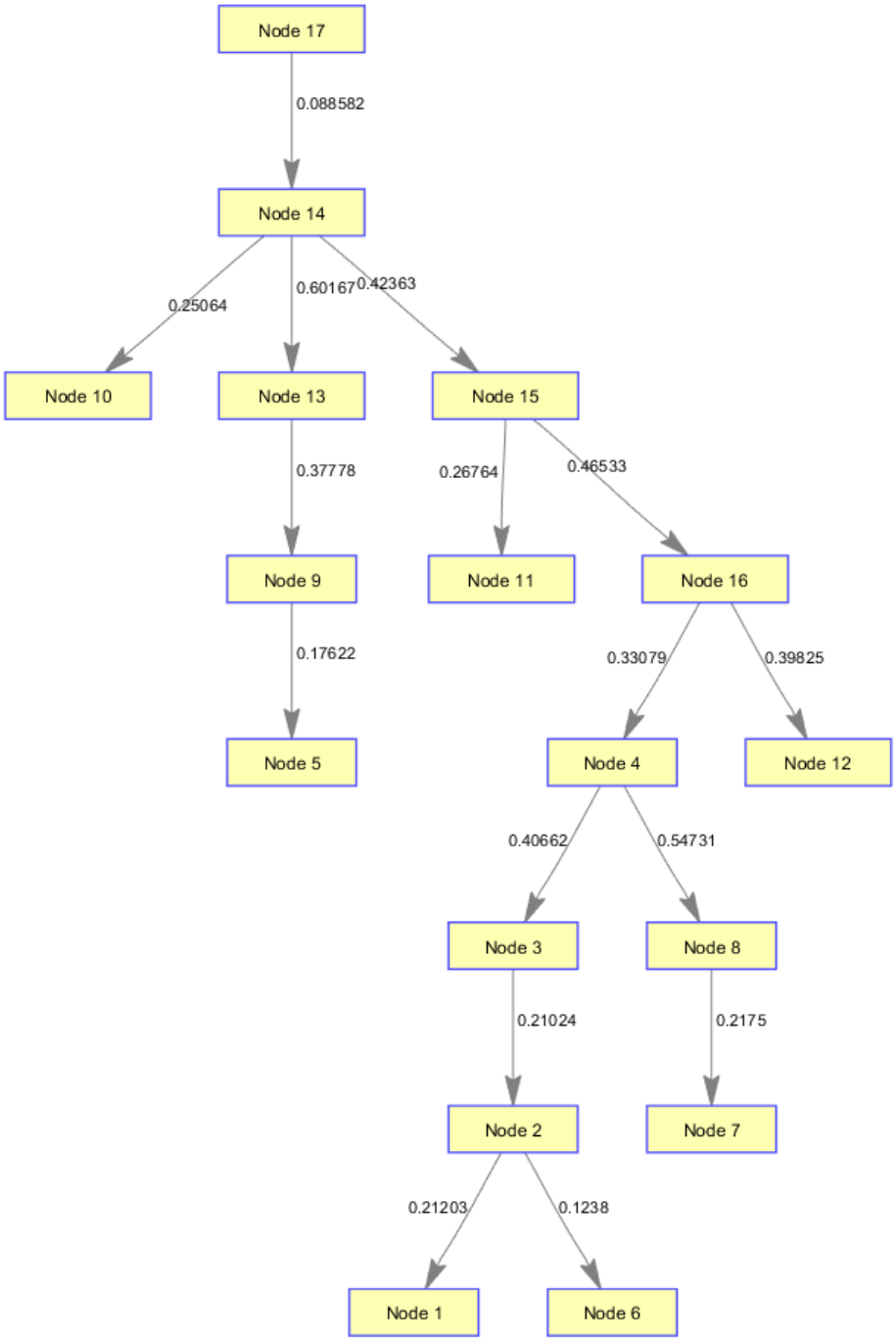
Win or Loss based classification accuracy:

The structure learnt was,



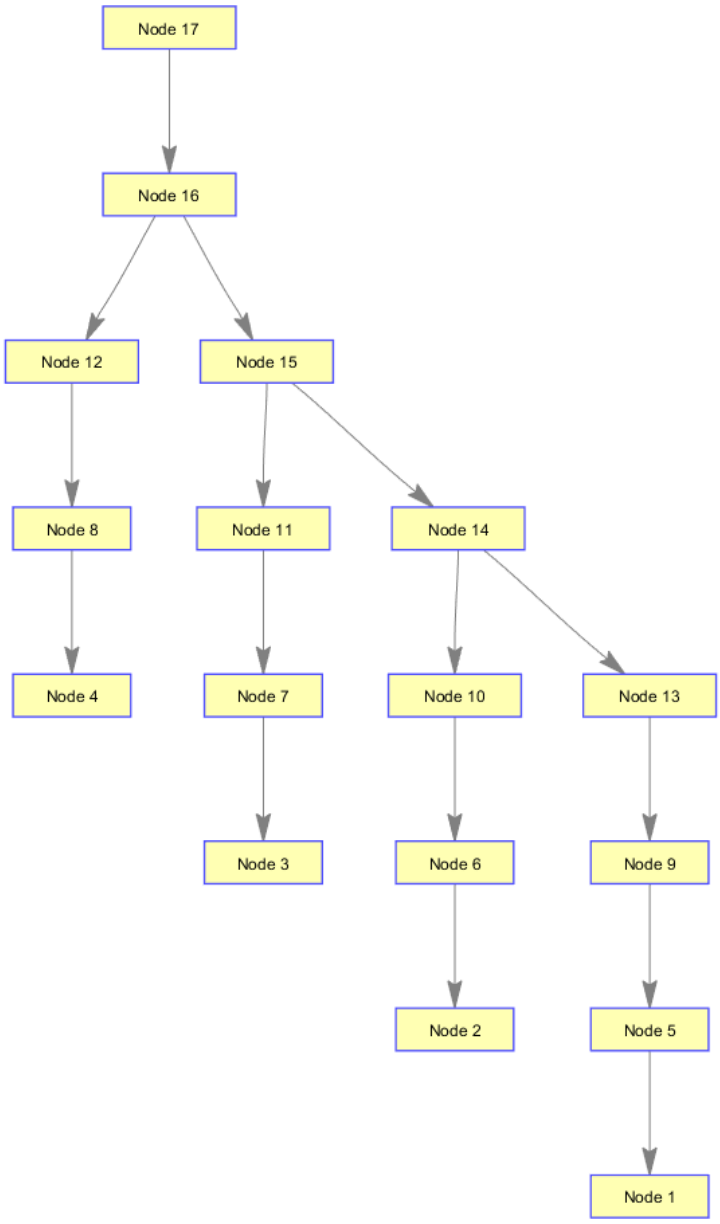
Accuracy: 64.19%

Move based classification: The structure learnt was,



Accuracy: 39.09%

Another Bayesian Network:

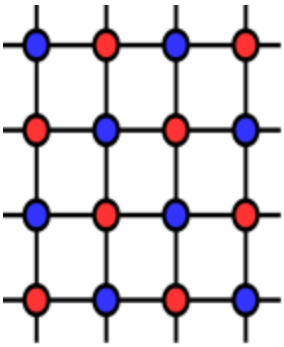


Accuracy:

Win or loss based classification: 64.19%

Move based classification: 30.15%

Markov Random Field:



Win or Loss based classification: 51.79%

Move based classification: 36.17%

Future Work:

More complex structure that captures the monotonicity of the tiles (i.e.,) the combination of the tile values is not affecting the label

Example: Use of Markov chain model

Data has to be preprocessed before training. All the symmetric state of the board should be converted to a same board state by taking transposes so that efficient model can be built

References:

1. https://en.wikipedia.org/wiki/2048_(video_game)
2. https://www.reddit.com/r/MachineLearning/comments/23n9pt/reinforcement_learning_vs_expectimax_for_a_2048_ai/
3. http://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048
4. http://www.utdallas.edu/~nrr150130/cs6347/2016sp/lects/Lecture_4_MRFs.pdf