Training a Minesweeper Solver

Luis Gardea, Griffin Koontz, Ryan Silva
CS 221, Autumn 2015

Abstract—**Minesweeper, a puzzle game introduced in the 1960's, requires spatial awareness and an ability to work with incomplete information. Utilizing different machine learning and artificial intelligence approaches, we implemented solvers that make use of linear and logistic regression, reinforcement learning, as well as constraint satisfaction problems. We were able to have various levels of success with different board sizes using different models, finding that the CSP model functions best, with the other models being limited by the difficulty of enumerating every board configuration for a given board size.**

I. Introduction

To start a game of Minesweeper, the player is presented with a rectangular grid of tiles, behind which are hidden a certain number of randomly distributed mines (the standard board is 16x16 with 40 mines). The player uncovers a tile by clicking it; if the clicked square is not a mine, it reveals an integer (its value), which is the number of adjacent uncovered tiles (including those that only share a corner) that contain a mine. Using this limited information, the player's goal is to uncover all of the tiles that do not have mines, but the player loses the game at any point if an uncovered square contains a mine. Using the techniques discussed in this class, we aimed to implement programs that can play Minesweeper.

Given a particular board state, such a solver must be able to decide which tile to uncover next, knowing only the information given by currently uncovered squares. Sometimes this information is sufficient to determine with 100% certainty that a mine will be in a given location; in other cases, one can only know the probability distribution of a mine's location. Thus, a solver must be able to find guaranteed "safe" tiles that cannot possibly contain mines, and when no such tiles exist, it must select the tile with the lowest probability of containing a mine. Notably, an early algorithm for this problem has already been written: by enumerating all possible mine configurations that satisfy the uncovered tiles' values, one can determine the probability that a mine is in any particular tile. However, the time complexity of this approach quickly makes the algorithm infeasible. Determining whether a given mine configuration satisfies the board constraints was proven to be NP-complete in 2000 [1]; because the previously mentioned algorithm involves enumerating all possible solutions to this NP-complete problem, it belongs to a more difficult class of problems called #P-complete [5]. Thus much of the difficulty in designing a Minesweeper solver lies in approximating the probability that a mine will be underneath any specific tile. The problem is further complicated by the fact that the tile with the lowest probability of containing a mine is not always the optimal move.

II. Literature Review

Several approaches have been made to solve games using machine learning. Mnih et. al. implemented a deep reinforcement learning technique used to learn strategy for playing Atari games [4]. A modified Q-learning algorithm was enhanced by function approximation with a convolutional neural network, which was able to effectively generalize learning of the state space. As for specific methods applied to solving the game of Minesweeper, several attempts were made in the 90's, where Adamatzky constructed a cellular automaton that populated an $n \times n$ board in $\Omega(n)$, with each cell having 25 neighbors and 27 states [2]. Various other models have been made, ranging from use of genetic algorithms, graphical models and other learning strategies [3]. There have been previous implementations to the CSP approach, which is the current "state of the art" method [3][8]. Nakov and Wei [5] derive bounds on the complexity of playing Minesweeper optimally. The authors formulate the Minesweeper game as a POMDP and use enumeration to convert the game to an MDP, while also reducing the state space. They then use value iteration methods to solve the MDP for a 4x4 board, but the method is not scalable.

III. Methods/Approaches

A note is to be made for the general approach with which the first move is made. Although initially, any random square will have the same probability of being a clear, given mine density d,

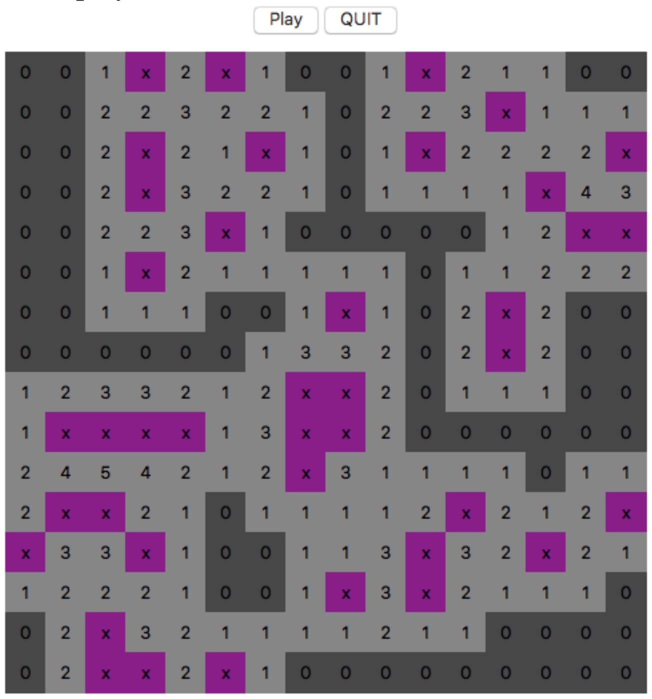
$$p(x_{ij} = \text{clear}) = 1 - d$$

however, in order to win the game, it is helpful to obtain a zero, as this will clear out a particular square and also all of the squares adjacent to it. We then have the following probabilities for different squares of being clear

$$p(x_{ij} = 0 | ij = \text{corner}) = (1 - d)^4$$
$$p(x_{ij} = 0 | ij = \text{edge}) = (1 - d)^6$$
$$p(x_{ij} = 0 | ij = \text{inside}) = (1 - d)^9$$

Because of this, if we select a corner square, we know that we will clear out a larger area of the board, gaining more information on the board and increasing the probability of winning.

Another note to be made is the fact that testing is being done with the standard real rules. This means that the player cannot lose on the first move, thus if the first move is actually a mine, the game is implicitly restarted and not counted as a game so that the move that the player selected is not a mine.



0	0	1	x	2	x	1	0	0	1	x	2	1	1	0	0
0	0	2	2	3	2	2	1	0	2	2	3	x	1	1	1
0	0	2	x	2	1	x	1	0	1	x	2	2	2	2	x
0	0	2	x	3	2	2	1	0	1	1	1	1	x	4	3
0	0	2	2	3	x	1	0	0	0	0	0	1	2	x	x
0	0	1	x	2	1	1	1	1	1	0	1	1	2	2	2
0	0	1	1	1	0	0	1	x	1	0	2	x	2	0	0
0	0	0	0	0	0	1	3	3	2	0	2	x	2	0	0
1	2	3	3	2	1	2	x	x	2	0	1	1	1	0	0
1	x	x	x	x	1	3	x	x	2	0	0	0	0	0	0
2	4	5	4	2	1	2	x	3	1	1	1	1	0	1	1
2	x	x	2	1	0	1	1	1	1	2	x	2	1	2	x
x	3	3	x	1	0	0	1	1	3	x	3	2	x	2	1
1	2	2	2	1	0	0	1	x	3	x	2	1	1	1	0
0	2	x	3	2	1	1	1	1	2	1	1	0	0	0	0
0	2	x	x	2	x	1	0	0	0	0	0	0	0	0	0

The image above is the GUI written for the CSP implementation. The purple squares with "x's" are the mines that were safely marked as mines. All other squares are the uncovered squares with their value shown.

A. *Baseline and Oracle*

In order to determine the difficulty of this problem, it is important to calculate the gap between the efficacies of a baseline and an oracle. For a baseline, we initially considered writing a program that plays by random moves. However, the probability of such a program winning rapidly approaches zero for even for small boards, and does not give us an accurate estimate for a baseline. Instead, we chose an average beginning player, who wins at a rate of 35% on an 8x8 board [5]. The selection of an oracle was more difficult. One possibility is to use an algorithm that cheats by looking up the location of mines, but this would achieve a success rate of 100% and again would not give a very accurate estimate of the upper bound. Truer oracles are difficult to implement though. The theoretical upper bound is a minesweeper agent that always behaves optimally, picking the guaranteed mines when possible, and selecting the probabilistically optimal choice when no tile is guaranteed to be safe. However, implementing such a program is the goal of this paper. Thus, we chose an oracle that can look up the location of mines a fixed ratio of the time, and other times must make a random guess.

B. *Supervised Learning*

Data for our supervised learning approaches is generated in the following manner. Each square can be represented as a feature with an integer determined by the squares state in the current board. If a square is uncovered, the corresponding feature is represented by the number of mines to which the square is adjacent. If the square is covered, it is represented by a COVERED value, and if the square is out of bounds, it is represented with an OUT_OF_BOUNDS value. We generate data by simulating games of Minesweeper, and play each game until it is won by always choosing a correct move. At each state, the perimeter consists of all squares that are covered, but are adjacent to uncovered squares. Moves are chosen randomly from squares within the perimeter with some fixed probability, and otherwise from all uncovered squares. This method models normal game play where most moves are selected from the perimeter of the current board. In our work, the machine learning algorithms do not flag squares indicating a mine is present; they only predict whether a square is suitable to be a next move. We play a version of Minesweeper that automatically uncovers neighbors of squares that are not adjacent to any mines. The general greedy algorithm for playing Minesweeper with a predictor is presented in Algorithm 1. Two approaches using supervised learning to solve Minesweeper are presented.

1) Local Classification: This method uses the local board configuration to classify an uncovered square in a Minesweeper game. The classifier uses a feature extractor to obtain input from the local board space.

Algorithm 1 playingMinesweeper

```
1: while not game over do
2:     Use trained model to predict all squares in the perimeter
3:     if No reasonable moves in the perimeter then
4:         Choose random move outside perimeter
5:     else
6:         Probe square with lowest chance of being a mine
7:     end if
8: end while
```

This feature extractor was chosen because nearby squares contain the most relevant information about the uncovered square that is currently being classified. Note that feature vectors in this dataset can potentially be correctly labeled in both classes, since the same board state can have multiple mine configurations. This means that the dataset is not separable, and to solve this non-separability, we label squares positively only when there is certainty that the square does not contain a mine (i.e there are no mines in a given square for all possible mine configurations).

Gathering data in this way tends to skew the data towards the negative class (the class predicting a mine is present). Sixty-six percent of our data is labeled negatively; this has an adverse effect because predictors trained on this data will favor making a prediction that a mine is present, and tend to misclassify safe squares. To counteract this, during training we give the positive class higher weight so that the classifier makes more accurate predictions about safe moves.

We train a logistic regression as well as an SVM classifier for predicting uncovered squares [7]. The new optimization problem for the SVM with weights β_i is

$$\min_w \Phi(w) = \frac{1}{2}w^\top w + C\sum_{i=1}^{m}\beta_i\xi_i$$

subject to

$$\begin{aligned} y_i(w^\top\phi(x_i)+b) &\geq 1-\xi_i, && i=1,\dots,m \\ \xi_i &\geq 0, && i=1,\dots,m \end{aligned}$$

Then, our dual becomes

$$W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j K(x_i,x_j)$$

subject to

$$\sum_{i=1}^{m} y_i\alpha_i = 0$$

$$0 \leq \alpha_i \leq C\beta_i \qquad i=1,\dots,m$$

and KKT conditions become

$$\begin{aligned} \alpha_i[y_i(w^\top\phi(x_i)+b)-1+\xi_i] &= 0, \quad i=1,\dots,m \\ (C\beta_i-\alpha_i)\xi_i &= 0, \quad i=1,\dots,m \end{aligned}$$

2) Global Probability Regression: This method uses all squares as features, and regression is done on the output of a full Minesweeper solver, which calculates exact probabilities for every square in exponential time. Both the training data as well as labels are high dimensional vectors in this regression problem, and the label is a real valued vector. The dimensionality of the dataset cannot be reduced without losing accuracy because of symmetries in the game and the fact that every square contains essential information for predicting probabilities.

For this problem, we use a multiple kernel ridge regression [6], with cross validation for setting the meta-parameters for this problem. Kernelizing the regular form of multiple kernel ridge can be done as follows.

The closed form of the parameters of the multiple ridge regression problem is given by

$$\theta^* = ((X^\top X + \lambda I_n)^{-1}X^\top)Y$$

where Y is a matrix of training labels. Using the identity

$$(B^\top R^{-1}B + P^{-1})B^\top P^{-1} = PB^\top(BPB^\top + R)^{-1}$$

we can reformulate θ^* as

$$\theta^* = X^\top(XX^\top + \lambda I_m)^{-1}Y.$$

To make a prediction with kernelization, we can use

$$\begin{aligned} \theta^{*\top}x &= Y^\top(XX^\top + \lambda I_m)^{-1}Xx \\ &= Y^\top(K + \lambda I_m)^{-1}k \end{aligned}$$

where

$$K_{ij} = \langle x_i, x_j \rangle$$

and k is a vector of inner products between the data and new x:

$$k = (K(x_1,x),\dots,K(x_m,x))^\top$$

C. Simplified Q-Learning

One approach involved modeling minesweeper as an MDP and using a modified version of Q-learning to discover the best actions for each given board configuration. After learning the Q values, the algorithm would then use the values obtained to play minesweeper. The standard Q-learning algorithm takes the following form:

$$\hat{Q}_{opt}(s,a) \leftarrow (1-\eta)\hat{Q}_{opt}(s,a) + \eta(r + \gamma\hat{V}_{opt}(s'))$$

This structure allows the algorithm to learn not just about the direct reward of a particular action, but whether a particular action is more likely to lead to reward in the long-term. While this is crucial for many games like chess, we are not as concerned with the endgame result in minesweeper. Instead we are more interested in the immediate reward: whether a particular move will uncover a mine on a specific board configuration. In order to approximate this reward, we remove the $\gamma\hat{V}_{opt}(s')$ term, leaving

$$\hat{Q}_{opt}(s,a) \leftarrow (1-\eta)\hat{Q}_{opt}(s,a) + \eta(r)$$

By letting the reward for choosing a non-mine be some positive number (say, 1) and that of choosing a mine be some negative number (-1), we are able to estimate which tile is least likely to have a mine in a given board configuration by finding the tile with the highest Q value. With this in mind, the initial implementation is shown below. While this approach had

Algorithm 2 initial Q-Learning

```
1:  Begin by probing a corner square
2:  while not game over do
3:      $s \leftarrow$ current state of the board
4:      Uncover square at random location $(a, b)$
5:      if square is mine then
6:          $r \leftarrow -1$
7:      else
8:          $r \leftarrow 1$
9:      end if
10:     $\hat{Q}_{opt}(s,a) \leftarrow (1-\eta)\hat{Q}_{opt}(s,a) + \eta(r)$
11: end while
```

moderate success, it was very slow at exploring the state space. This led us to wonder if we could gather information on possible actions more quickly. We devised a further modification that at each state would update information about all correct moves, but would only choose one to proceed. The algorithm is shown in the following column. This training algorithm makes a few changes. One change is that the algorithm

Algorithm 3 improved Q-Learning

```
1: Begin by probing a corner square
2: while not game over do
3:     $s \leftarrow$ current state of the board
4:     $Array \leftarrow$ all tiles on frontier not mines
5:     for tile in $Array$ do
6:         $\hat{P}(s,a) \leftarrow \hat{P}(s,a) + 1$
7:     end for
8:     probe random square in $Array$
9: end while
```

only considers correct moves. In addition, we have introduced the notation $\hat{P}(s,a)$ because we no longer store the Q value for a given (state, action) pair; in fact $\hat{P}(s,a)$ more closely resembles the probability that a given tile does not contain a mine. Removing the η terms is only acceptable because we do not compare P values between different states; we only compare different actions within the same state. Suppose our learning algorithm has visited a state S n times. Tiles with a P value close to n have very rarely contained mines, and tiles with a P value closer to zero have frequently contained mines. Thus, tiles with higher P values in a given board state are less likely to contain mines than tiles with lower P values in the same state. This assumes that S is large, however, in order to acquire a proper probability distribution. After storing P values for these (state, action) pairs, we must use them to play minesweeper and determine our win rate. For each move, the playing algorithm plays the tile from the frontier with the highest P value. However, the only exception is if all tiles from the frontier have a P value of zero, and if the training algorithm had seen this state at least once: the implication being that there may not be any correct moves on the frontier. In this case, the playing algorithm selects a random tile that is not on the frontier and continues.

D. Constraint Satisfaction Problem (CSP)

When posing the problem of solving a Minesweeper game as a CSP [8], all board positions are thought of as boolean variables, having a value of either 0 or 1 to represent the presence of a mine. When a tile is probed, either a mine is found, in which case the game is lost, or its value is shown. Knowing the value of a tile allows for a constraint to be set on the variables that surround it; this is done by stating that the sum of these variables must equal the value of the tile. Thus, our constraint is

$$x_{ij} = \sum_{k,l} x_{kl}$$

where x_{ij} is now a known constant and where $k \in \{i+1, i, i-1\}, l \in \{j+1, j, j-1\}$ and are within the bounds of the rows and columns of the board. When new values are discovered, the constraint is simplified by subtracting from both sides. Variables can be simplified further when they share variables in common; given the constraints $x_1 + x_2 + x_3 + x_4 = 2$ and $x_1 + x_2 = 1$, it can be deduced that $x_3 + x_4 = 1$ and the larger constraint can be discarded. Constraints can be easily solved if they are trivial, ie. $x_{ij} = 1$, or if the constant is equal to the number of variables or 0, in which case all variables are mines or all are clear, respectively. This is an implementation of the Equation Strategy [8].

After all simplifications are done, if no safe move exists, we divide existing constraints into sets of constraints that have common variables and run a backtracking algorithm that finds all possible solutions. Each solution is grouped by the number of mines it requires and for each variable, a tally is kept of how many solutions require it to be a mine. The number of mines remaining and the number of mines a solution requires are used to throw out infeasible solutions. If after all solutions are computed, a square is found to be a 0 or 1 in all of the solutions, that square can be probed or marked a mine, as its value is then certain.

There are cases, specifically when all of the solutions to a coupled set of constraints require the same number of mines and none of the variables in the set of constraints have neighbors that are part of another constrained set or unknown. In these cases, there is at most a 50 percent chance of guessing correctly and no new information will help in determine with certainty. Because of this, the algorithm makes the guess sooner rather than later, since it may still guess incorrectly in the future, so in order to save time, it should guess now. No information is learned from these “crap shoots”, even in the case that the guess was correct. After this, another guess must be made to decide on where to probe. The best guess is calculated by summing up the number of solutions where a variable has a value of zero and dividing it by the number of solutions for that variable. There is also an estimate made on the probability of there not being a mine on the unexplored and unconstrained areas of the board. If the probability of there not being a mine is greater in the unexplored area, then we pick, in order of precedence, first, if there are unconstrained corner squares, we probe one, if not, then we attempt to probe unconstrained edge squares, if there are none, then we finally attempt to probe a random square that is unconstrained; this goes in accordance to what was stated above, where a square with a zero is the desired value.

The backtracking algorithm used for the CSP attempts to find all of the possible solutions to satisfy the constraints. Variables are assigned values, with variables being assigned first if they are more constrained. Each variable is tested with a 0 first, then a 1. Each assignment is checked for consistency, if one assignment is inconsistent, the other is tested, if assignment doesn’t work, then backtracking occurs. When all variables are assigned, a solution is found and the algorithm backtracks to find the next solution.

IV. Results and Analysis

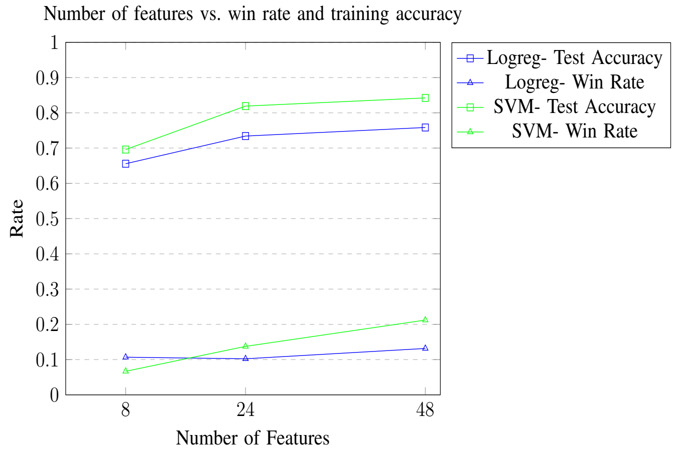
We use three metrics to understand how well our approaches perform in playing Minesweeper: the testing accuracy, which is how well the classifier predicts individual board states, the average percentage of the board uncovered, and the average number of game wins. We plot these three different metrics because win rate alone can be misleading: predictors with a low win rate may still achieve reasonable training and testing accuracy. This is because the machine learning and Q-learning algorithms must make many consecutive correct predictions in order to win; one can imagine a scenario where such an algorithm is able to clear a large percentage of the board before making an incorrect move. In this situation, the average percentage of the board uncovered might be a more revealing statistic than win rate alone.

First, we observe the relative performance of the four approaches. The classification, linear regression, and Q-learning algorithms all exhibit moderate success on 4x4 boards; the Q-learning approach achieves roughly a 70% win rate, the greatest of the three. However, the performance of these three algorithms drops immediately as board size increases. Notably, each has a winning rate of <10% on a 5x5 game board, and <5% on a 6x6 board. The CSP, however, enjoys significantly greater success, with a win rate of approximately 80% on an 8x8 board. Furthermore, its win rate decreases very little as the board size increases, seeing a win rate of slightly less than 70% on a 32x32 board.

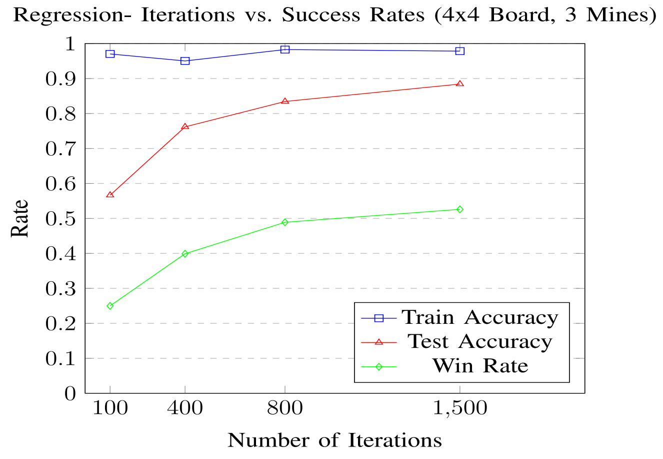
Second, we would like to highlight the decrease in win rate as mine density increases. This effect applies to all the algorithms – a game with more densely placed mines is simply harder to win because a game with more mines has far more possible layouts. Consider an nxn board with p mines, where $p \leq n^2$. On this board, the number of distinct board configurations is equal to $\frac{n^2!}{p!(n-p)!}$ Excluding rotations and symmetric configurations, this value drops to $\frac{n^2!}{8p!(n^2-p)!}$. One can see that the number of possible mine configurations is maximal when $p = n^2/2$; not surprisingly, this number increases as p approaches $n/2$. A greater number of possible mine configurations causes a more challenging game simply because there is less certainty. It results in a larger state space, and there are fewer guaranteed safe moves.

While the Q-learning model shows promise, it suffers from a hugely large state space. To approximate its magnitude, on a minesweeper board, we see that each tile can hold one of ten values: unknown (i.e. covered), 0 (i.e. empty), or 1-9. Thus we see that 10^{mn} is an upper bound on the number of board configurations. While this number is significantly greater than the true number of configurations (due largely to inconsistent configurations), the number still illustrates that the state space explodes as board size increases. The issue of increasing magnitude of state space is compounded by the fact that in order to be accurate, the Q-learning algorithm must visit each state many times. To see why, we recognize that the algorithm aims to compile a probability distribution of whether each tile on the frontier is or isn't a mine. Given the same board state (what is visible to the player), however, a tile may be a mine under one mine configuration, but be safe under another; in other words, different mine distributions can produce the same board state. Therefore the algorithm must visit each board state under a random sampling of mine configurations in order to accurately approximate the probability of whether each tile is safe. This is why the algorithm struggles so mightily with increasing board sizes; it must play significantly many more training games to achieve the same exploration rate of each state. As an example, when moving from a 4x4 to a 5x5 board, the state space can be said to increase by a factor of $\frac{10^{25}}{10^{16}} = 10^9$. According to this naive calculation, the algorithm must train for one billion times longer In order to achieve the same exploration level of each state. Again, the true value is smaller, although it still grows at an alarming rate. The enormously large, and rapidly increasing, state space of minesweeper explains the performance of the Q-learning algorithm. It requires far more training iterations than classification and linear regression because it needs to thoroughly explore the entire state space–it cannot generalize to unseen states. And its win rate drops quickly as board size increases because the state space simply explodes, and it cannot thoroughly explore larger boards in any reasonable amount of time.

The most significant observation about the classifier model is that it is highly prone to be biased when the feature extractor operates on a relatively small amount of the board. We vary the number of local features used in training to illustrate how the number of features affects bias in the classifier as well as the win rate of the algorithm.

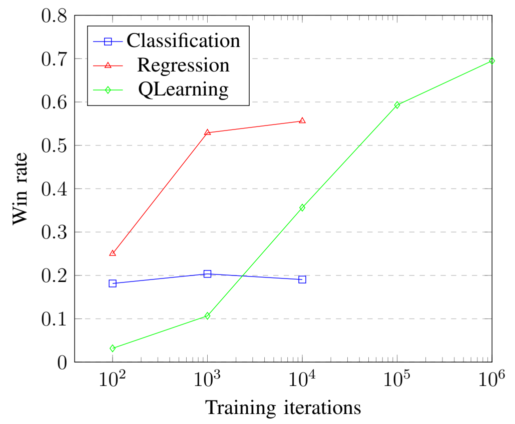


The regression model suffers from high variance and choosing the correct parameters for regularization is essential for good learning. We illustrate how test accuracy (the accuracy on individual states) as well as win rate increase with the size of training the training set, correlating less variance with higher win rate.

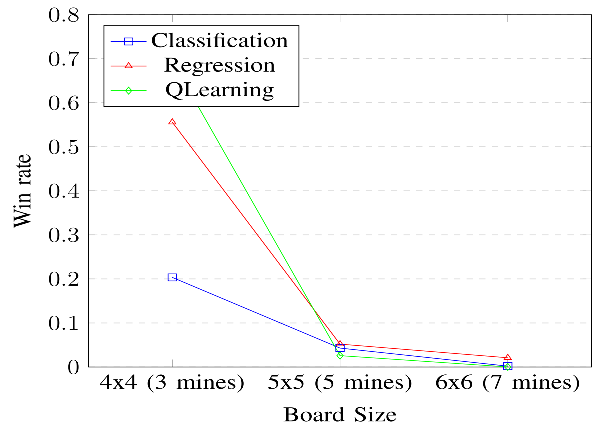


Supervised learning methods each have benefits and drawbacks. The classifier model has potential to scale to larger size boards, since the feature vector is not dependent on the board size, and the prediction is made solely on the local board state. Yet this approach currently underperforms compared to our other two machine learning approaches. On the other hand, because we use an exponential time Minesweeper solver to train the regression model, we cannot train our algorithm on large boards. Therefore, we optimize learning on smaller boards where learning the probabilities for every square is feasible.

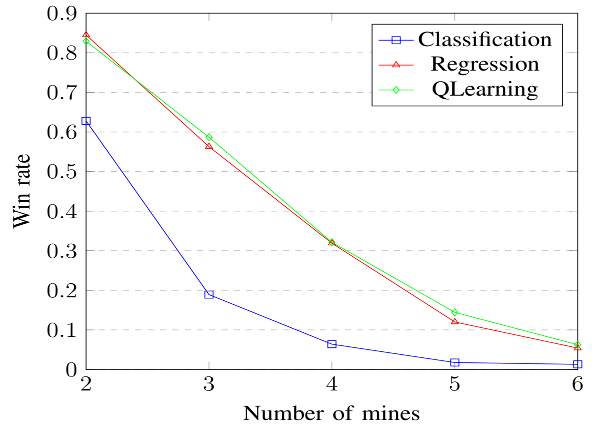
Number of training iterations on a 4x4 board with three mines vs. Win rate



Board size vs. Win rate

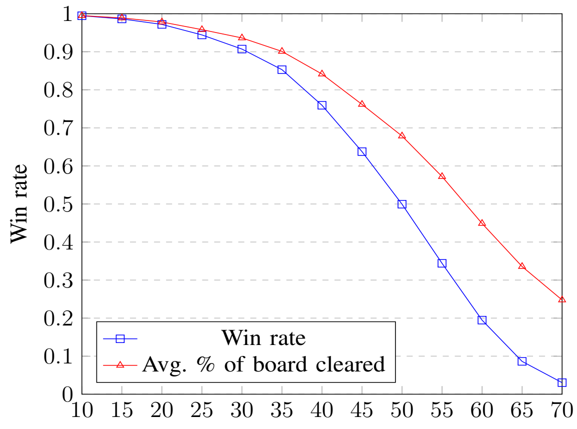


Number of mines on 4x4 board vs. win rate



The CSP approach was the most successful at correctly solving larger boards. This is due to the fact that it does not have to enumerate every possible state in order to obtain good results. The way that the problem is posed allows the CSP to make as few guesses as possible and most of the moves that it makes are done with full certainty of correctness. The guesses that the CSP solver does have to make come from the situation where it is forced to make a "crap shoot" guess; this is unavoidable and no optimizations could help the solver. The other type of guesses comes when there is not enough information on the board to be able to solve systems of constraints.

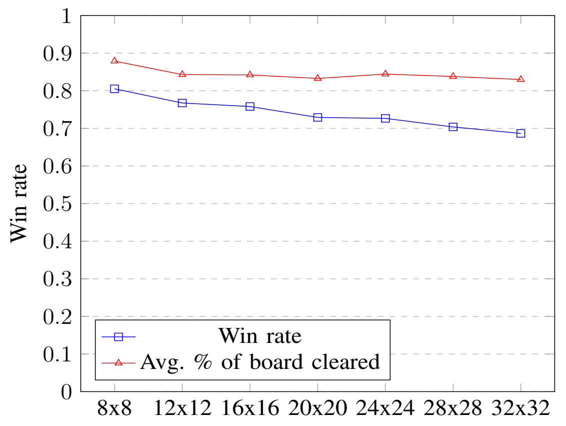
CSP: Number of mines on 16x16 board vs. win rate



Our results show that increasing mine density quickly drops the rate of success for the CSP solver, similar to the other solvers. Because there are more mines on the board, the likelihood of incorrectly guessing for mines greatly increases and there are also fewer non-mine squares to get information from. However, while the mine density greatly affects the CSP, the board size does not seem to affect it as much. The solver was tested using the same mine density (10 mines for an 8x8 board) on larger board sizes and, while the win rate did drop, it was by less than 15%.

While the CSP is more accurate than the other solvers for playing games on large boards, it is much slower. The other methods use most of the time in doing pre-computations and learning, but once that is done and saved, they can play many more thousands of games than the CSP, even when the CSP required no pre-computation. This is mostly due to the slow backtracking algorithm, which could be optimized to be faster.

CSP: Board size vs. win rate at constant mine density



V. Further Research/Work

Moving forward, there are several additional approaches to explore related to our current research.

The first is in improving the success rate of our CSP approach. While the CSP excels at determining which tiles are guaranteed to be safe and which tiles are guaranteed to be mines, it does not have an effective mechanism for making guesses in cases without complete certainty. Since our other approaches attempt to approximate mine probability, we believe that integrating them with the CSP will lead to a greater success rate. For instance, we propose training our Q-learning algorithm on a smaller board (say, 4x4 or 5x5) and applying it to certain regions of a larger board. Similarly, we also would like to utilize the classification algorithm's probability estimates to better inform the CSP's guesses, since this approach is inherently scalable. The resulting insight into which squares are least likely to be mines will help the CSP make more educated decisions. As stated in the above section, another area where the CSP could be improved is in performance; the CSP backtracking algorithm is slow and could be optimized to be faster.

In addition to improving the CSP's guess accuracy, we also believe that the approach of deep Q-learning holds promise. A key feature of Minesweeper is that the same method for solving one board state can directly be applied to another board state, which makes function approximation in Q-learning especially ideal. Specifically, given the strong local dependencies of making decisions while playing the game, we think using a Convolutional Neural Network (CNN), similar to [4], is a promising area of exploration because learning local strategies can be generalized to the entire board, greatly helping to learn the expansive state space for larger boards.

References

[1] Kaye, R. (2000). Minesweeper is NP-complete. *The Mathematical Intelligencer, 22, 9-15.*

[2] Adamatzky, A. (1997). How Cellular Automation Plays Minesweeper. *Applied Mathematics and Computation, 85, 127-137.*

[3] Maznikova, M. A. Minesweeper Solver.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *CoRR.*

[5] Nakov, P., & Wei, Z. (2003). MINESWEEPER, #MINESWEEPER.

[6] Welling, M. Kernel Ridge Regression *University of California, class notes.*

[7] Yang, X., Song, Q., Wang, Y. (2007). A Weighted Support Vector Machine for Data Classification. *International Journal of Pattern Recognition and Artificial Intelligence, 21(05), 961-976.*

[8] Studholme, C. (2000). Minesweeper as a Constraint Satisfaction Problem. *Unpublished project report.*