CS 4341 Project 3 Report
Robert Dutile & Myo Min Thant

1. Features:

Firstly, there were of course the required features, the owner of the bottom left position and the player who has the most tokens in the center 3 columns. Besides the two required features, we also extracted bottom_right, center_two and higher_chance. Bottom_right is basically just a mirror of bottom-left, just checks the value of the bottom left board position and outputs that. It just seemed to be about as useful as bottom left might be, so it was a natural addition. Center_two checks who owns the two most-centered positions of the board, and gives either the number of the player who owns the most positions, or 0 if either/neither owns them, or if both own one. These positions are the most strategically valuable on the board just by dint of how many ways there are to win from these positions, so control of them grants an advantage. Finally, the most complicated feature was our higher_chance feature. It essentially calculate the heuristic for each player, and scans through the board for occupied positions, adding an amount to the count of the owning player equal to the number of ways to win from that position. For example, if player 1 occupied bottom right position, the value of that position would be 3 since there are only 3 ways to win from that position: one horizontal, one vertical and one diagonal. If he occupied most-centered bottom position, the value will be 7: one vertical, two diagonals and four horizontals. These positions values are added up and compared to determine which player has higher chances to win. The program produces the number of the player with the highest score, or 0 if it is a draw.

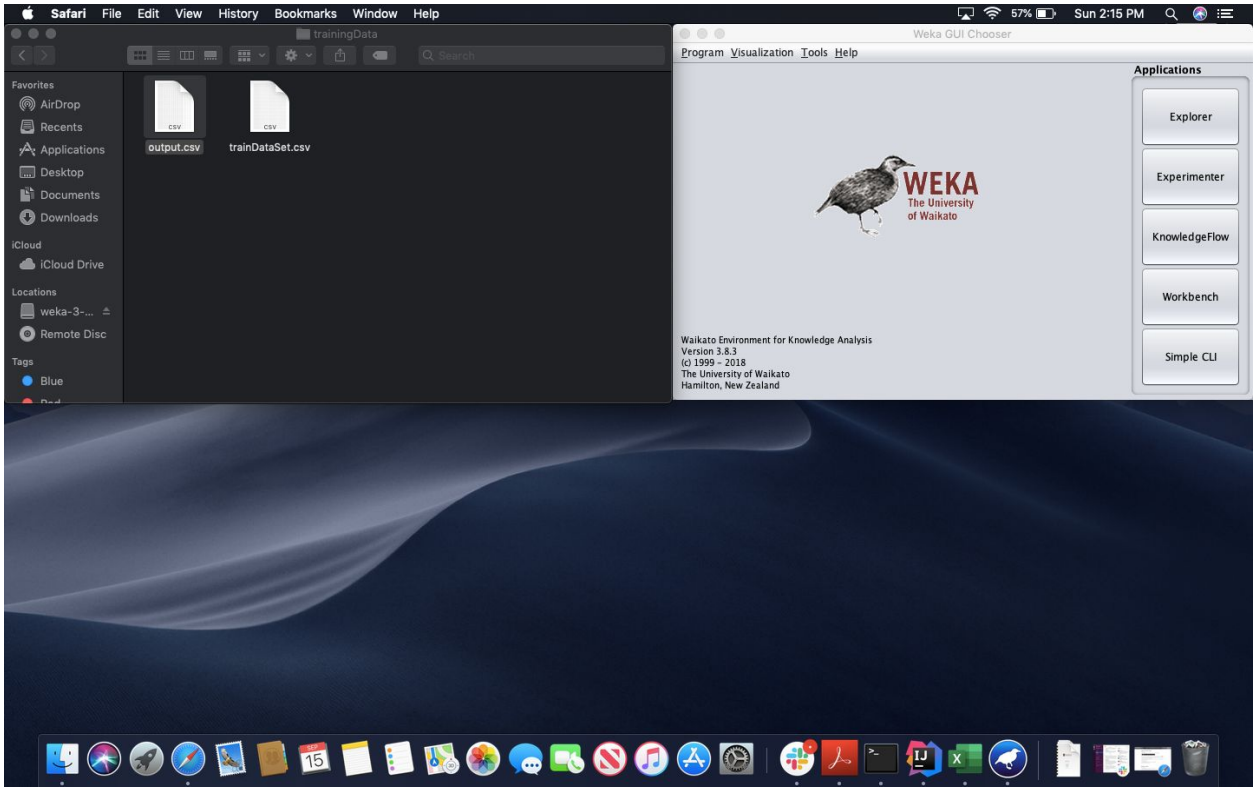
As far as the decision tree is concerned, the most important feature appears to be higher-chance, followed by center, bottom_left, center_two, and finally bottom_right.

2. Steps and methodologies

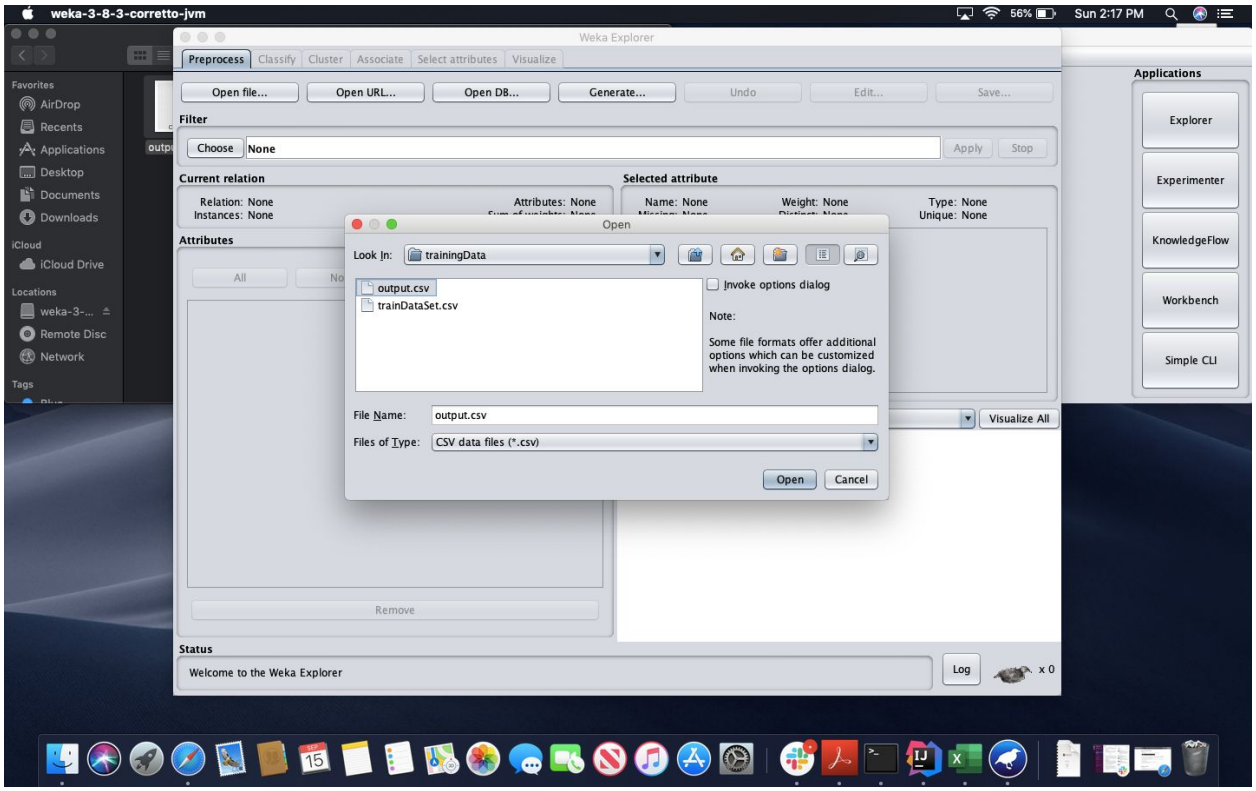
Since we used Java for extracting features from input file, we tested our decision trees with WEKA as suggested in the project description. We decided to test our decision tree using decision tree pruning and 10 folds cross validation. The results show pretty promising as they returns 79.2 correctly classified instances and achieves a precision of 0.786.

First we researched our features that could highly impact the decision tree. Then, we output the data to our output file. Using that output file, we opened it up inside WEKA. The first step would be to pre-process that data. Since our output file had 1st to 42nd columns as board configuration, we removed those attributes inside WEKA and left out winner, bottom_left, center, bottom_right, higher_chances and center_two. The next step would be normalized our data to nominal value. After that we started classification testing pruned and unpruned J48 decision tree and random forest tree with 10, 100, and 1000 trees.

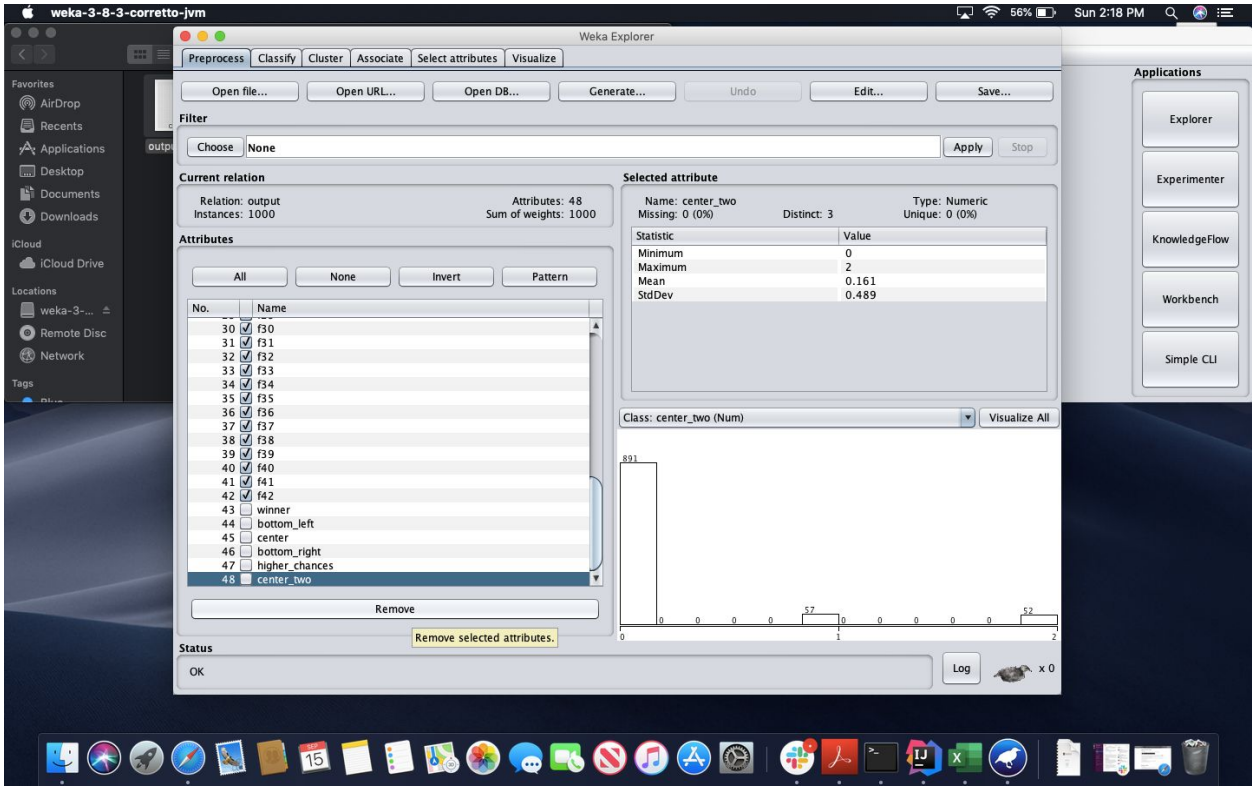
Below are the step by step procedures provided with screenshots.



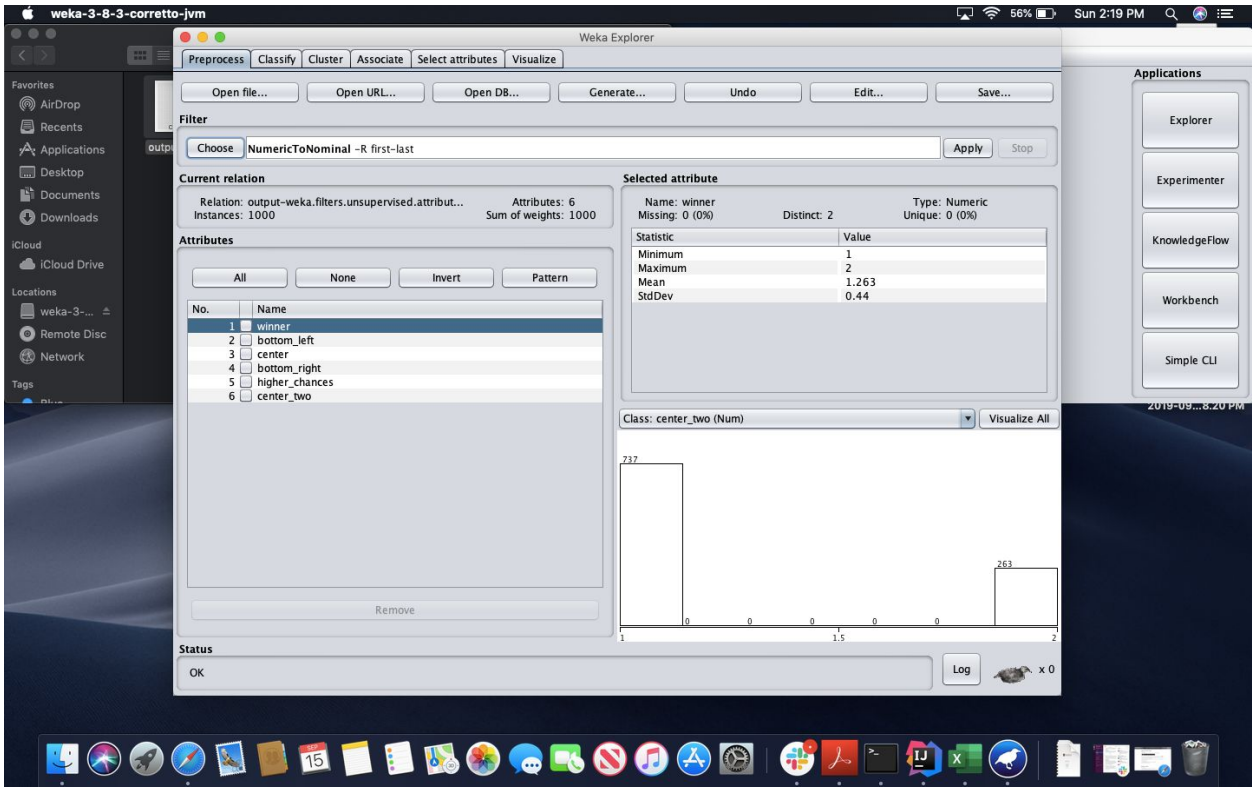
First, we opened up WEKA and chose “Explorer”. There is also output.csv file ready for testing.



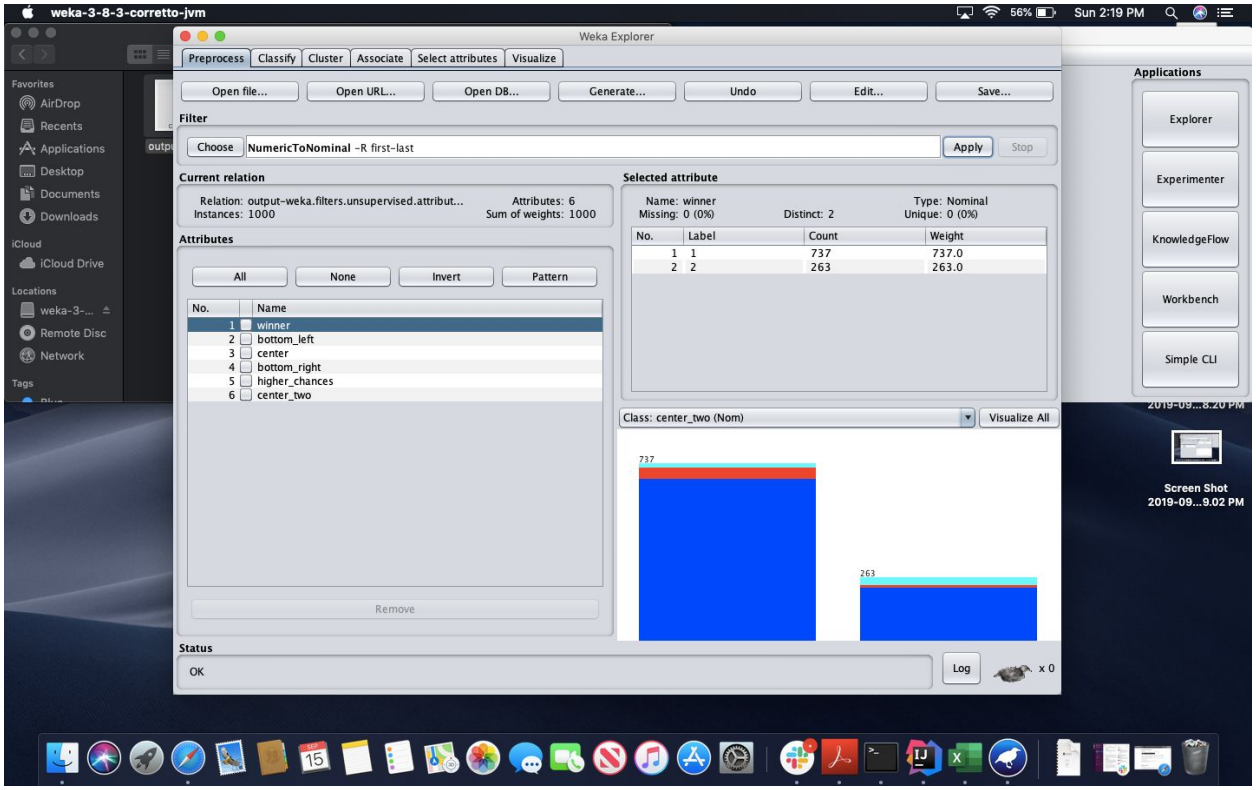
We chose “Open file…” and chose our output.csv file.



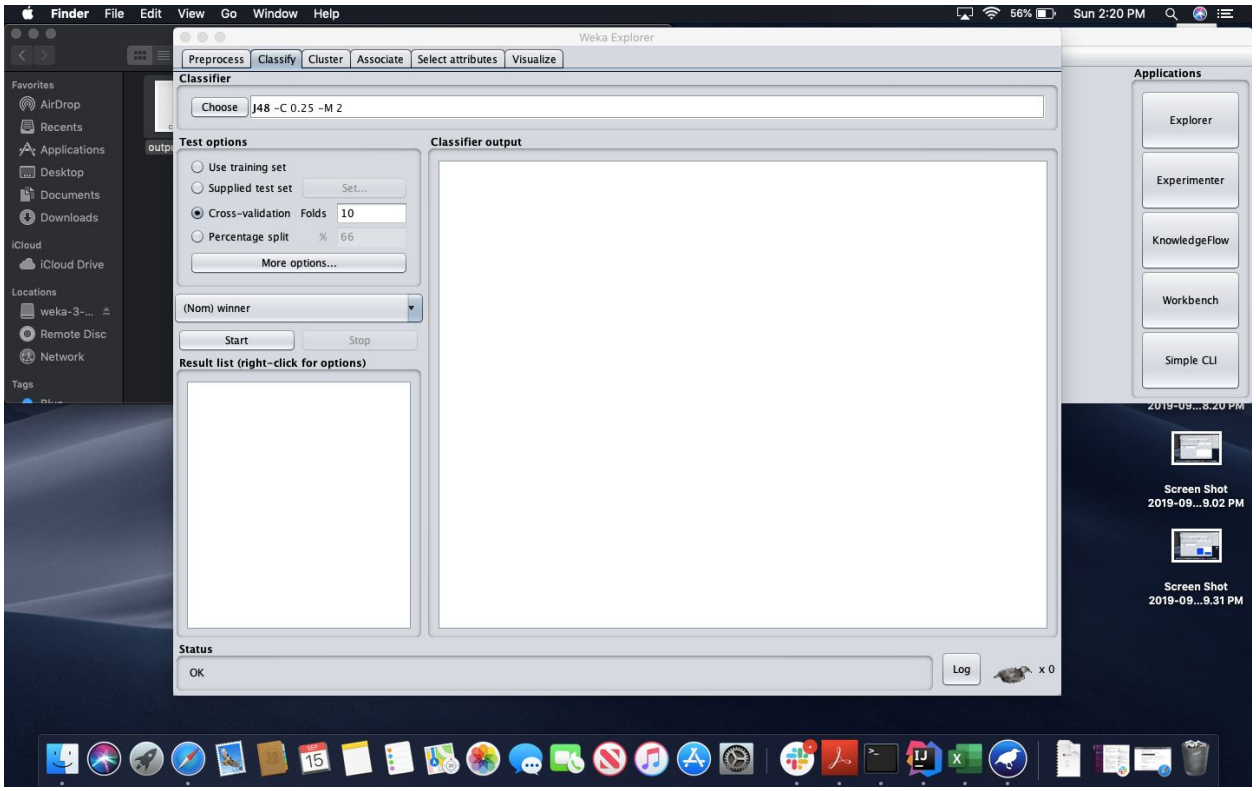
Then, we selected all the columns related to board configurations and removed them.



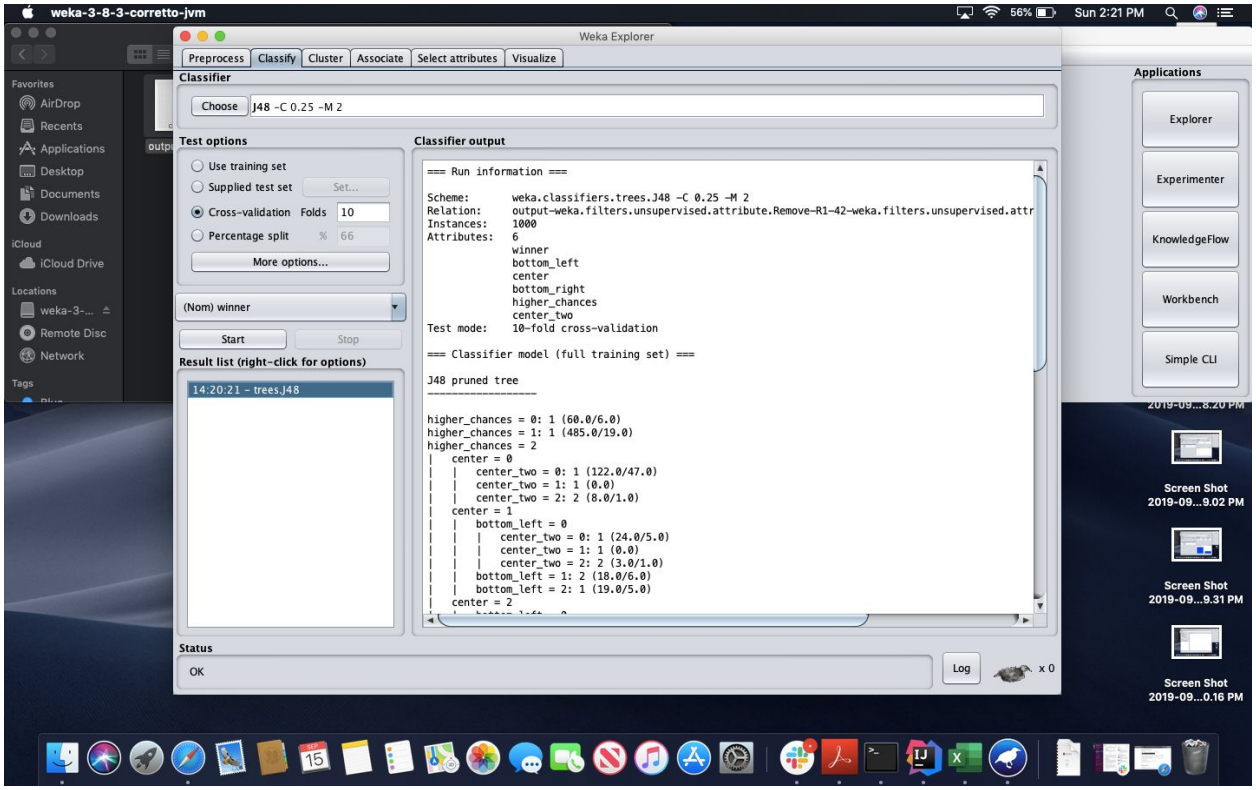
The next step would be filter our features into nominal value. Under “Choose” button, we selected “unsupervised” > “attribute” > “NumericToNominal”.



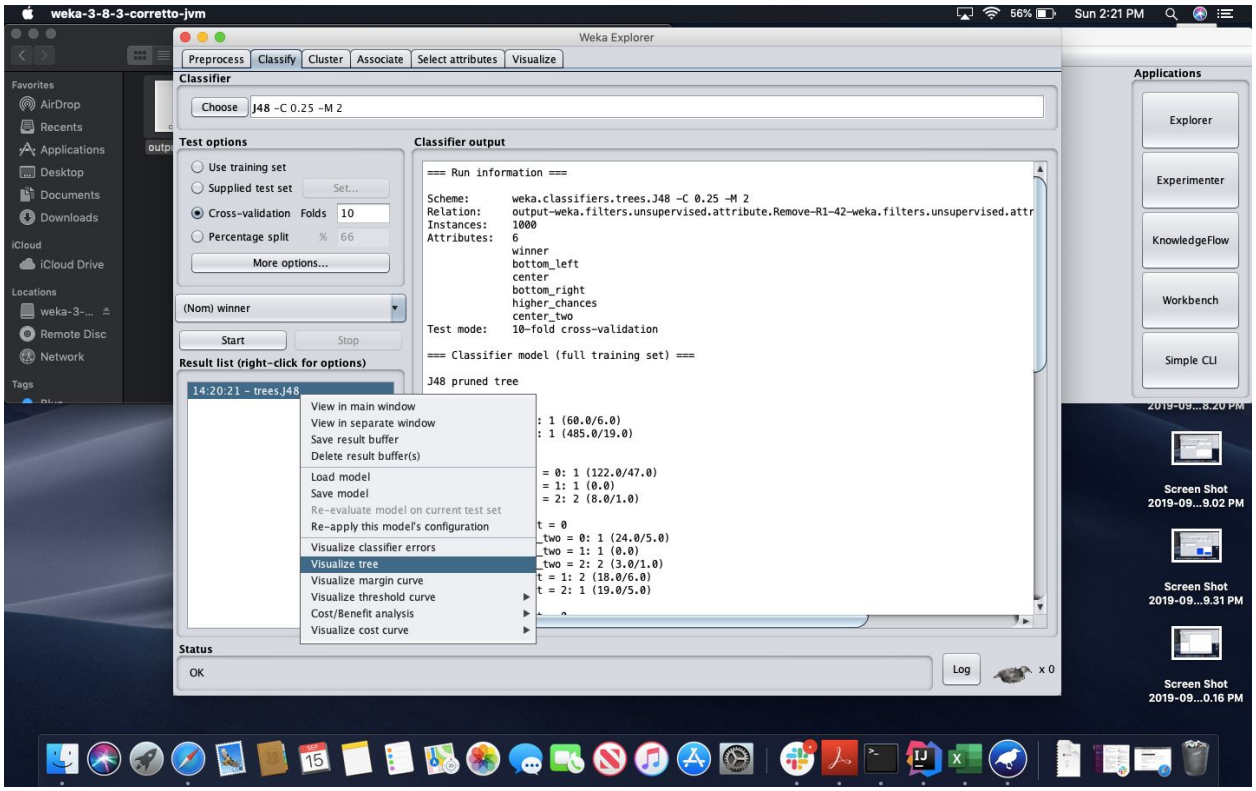
Then we could see that our attributes has type nominal instead of numeric.



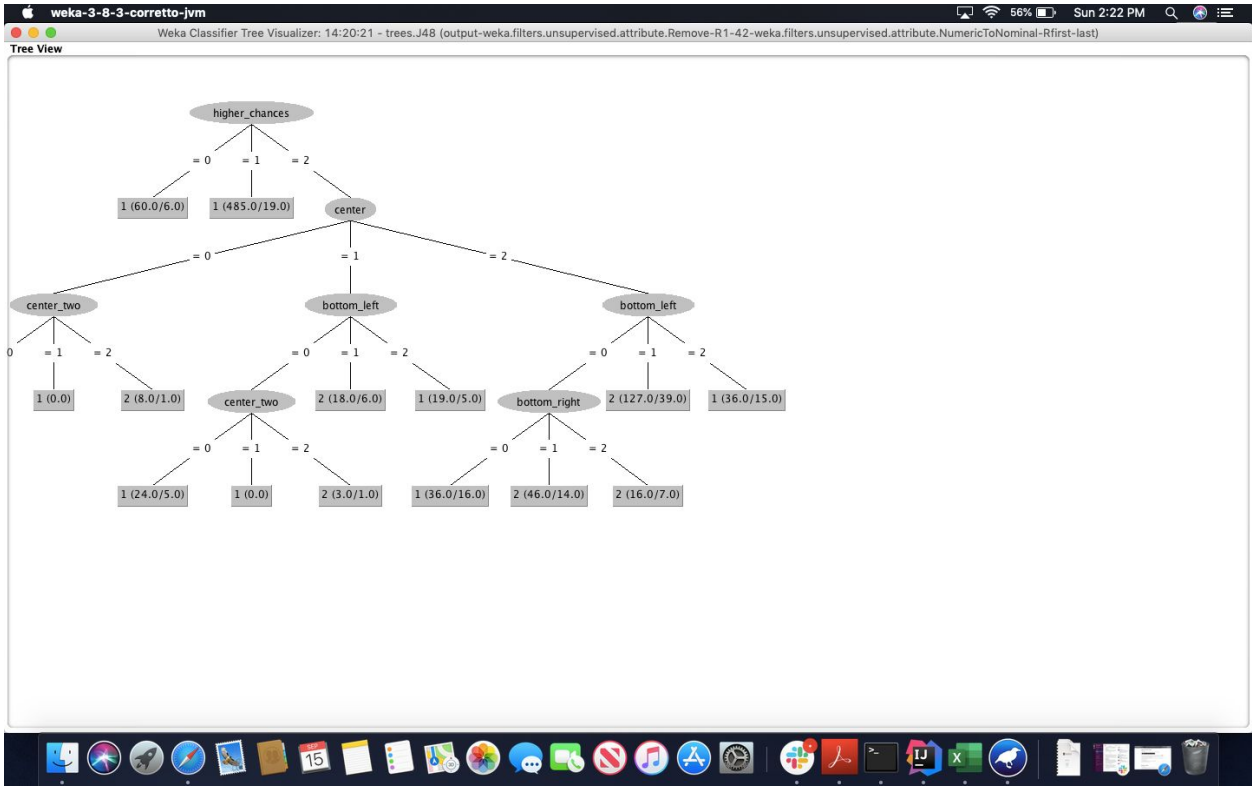
After pre-processing data, we went to “Classify” and chose J48 under classifier. We gave 10 folds for cross-validation, selected our target attribute as “winner” and started testing.



Then we will get the results.



To view the tree, right-click on our tree result and select “Visualize tree”.



The figure above is our J48 pruned decision tree with our extracted features.

3. Further testing

One of the testing we compared was to notice differences in results between using decision tree pruning and not using it. When we used decision tree pruning, the tree size became relatively smaller and classified slightly better than unpruned tree.

Decision Tree	Pruned	Unpruned
Tree Size(leaves count)	22(15)	37(25)
Correctly classified	792(79.2%)	787(78.7%)
Incorrectly classified	208(20.8%)	213(21.3%)
Precision	0.786	0.780

Another testing was comparing the random forest tree with different number of trees/iterations. Default WEKA value is 100 but we also tested on 10 and 1000 to notice if there was any significant difference. If we look at the comparison below, we don't see any significant difference among the three different random trees. Only noticeable result is that random tree correctly classified more with larger amount of trees but only by a slight difference.

Random Tree	10	100	1000
Correctly classified	780(78%)	781(78.1%)	784(78.4%)
Incorrectly classified	220(22%)	219(21.9%)	216(21.6%)
Precision	0.776	0.774	0.778

From the above two testing methods, we can also compare between using random forest tree and decision tree. When using decision tree pruning method, the returned results are a lot better by 0.8% than random forest tree using 1000 trees considering our dataset is pretty small with only 1000 configurations.

One final testing method we tried is using leave-one-feature-out. For this method to be effective, we left out the best feature in one trial and the worst feature in another trial from the above decision tree pruning. Again, we tested these two trials using decision tree pruning. Our best feature is higher_chances and worst is bottom_right. From the results below, we noticed that leaving out the best feature gave us a worse score than the full-featured decision tree. It went down by 3.5% in correctly classified instances. On the other hand, leaving out the worst feature improved by 1%.

	Leave out higher_chances	Leave out bottom_right
Correctly classified	757(75.7%)	802(80.2%)
Incorrectly classified	243(24.3%)	198(19.8%)
Precision	0.739	0.802