Introduction

For this practical session, we used decision trees to predict StarCraft players based on their keystrokes. We used both Knime and Python to build the decision trees. Our data set was collected data from the game StarCraft, which contained many files of varying sizes of data. This was the first variable we handled in creating our predictions. Decision trees work best when they have a lot of data. The StarCraft was enough for us to create a good classifier.

The goal of a decision tree is to predicate an outcome based on certain classifications of the input. It does this through creating a tree structure where it can create multiple paths that are filtered based on a classification. This supervised learning method was not the only method we used but ending up having the best results.

We started with the iris data set, then moved to the StarCraft data set. All the StarCraft data sets were created from a set of 955 professional games according to two parameters. The filenames gave us these parameters: the t first seconds of the game for which we count the key pressed, while theta gave us the minimal number of instances a player must have to be included in the data set. Using these parameters we were able to parse the data set and create our decision tree for each file. Within each file contained race, status, user, and a count of all keystrokes used. From this, we used all the keystrokes to classify our decision tree in order to be able to predict user, race and status.

Knime versus Python

We did both the Knime version and Python version for iris and StarCraft data sets. One reason is that we had trouble converting objects to codes, the other is that we needed to have some results in the Knime to prove the results we got from Python were correct.

For iris data set, we manage to build the Knime workflow for iris data sets easily by following the guidelines we were provided. Then we created the decision tree model in Knime. Our workflows are given in the attachment. After that, we manage to build our Python version for the iris data sets with the help of the teacher. Finally, we get a good-looking decision tree model print. You can see in Figure 1.

We tried to build a work flow for StarCraft, we use a loop to read all files, and get the score, but we did not manage to plot everything we wanted. During the second course, the teacher taught us to build our workflow on Knime step by step. We solved problems we met before successfully. You can find that in attachment. And Figure 2 is the picture we plotted about the relation between game count threshold and accuracy.

In fact, after writing the Python version for iris data sets, we can easily write the same for the StarCraft data sets. And let me show you in the next section.

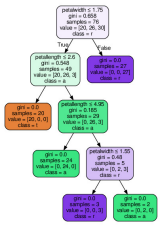


Figure 1: The model we trained by python. You can see the decision tree model we got through training. And its Gini value was 0 in leaves, meaning our model is very good.

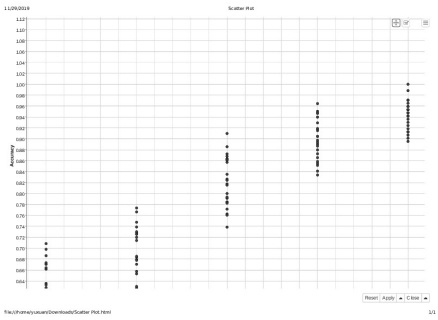


Figure 2: The relation between game count threshold and accuracy

Python version

In this part, we will show what we get from Python. At first, we manage to train the decision tree model. After that, we split data into train set and test set. Then, using the former to train and the latter to test if this model was good. And we get accuracy 0.95, which is good.

Finally, the structures of iris and StarCraft are very similar, the difference was that we need to add a loop to read all files and then process.

Figure 3 is one of many models we get.

Figure 4 is the relationship between game count threshold and accuracy. As game count threshold increase, the accuracy also increases.

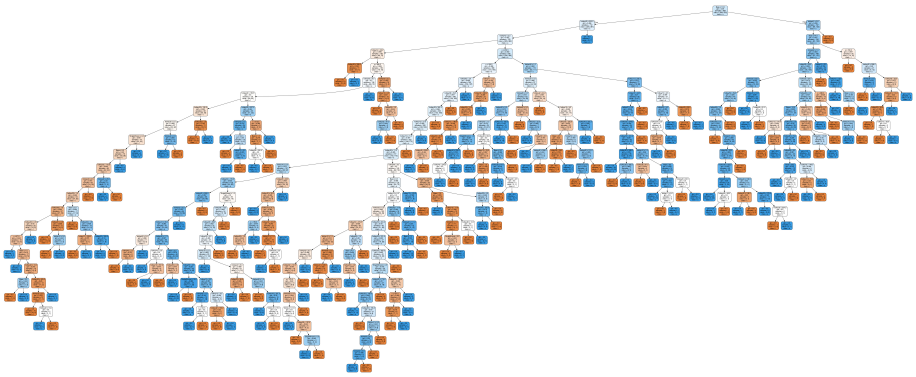


Figure 3: The decision tree model

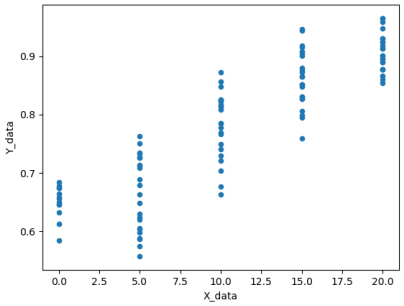


Figure 4: Relation between game count threshold and accuracy we get from Python

Now that we had a good model for all the StarCraft files. It was clear that the relationship between accuracy and game threshold was just a matter of how much data was provided. From this we decided to compare more relationships by adding another dimension: game time. In Figure 5 you can clearly see the same relationship we had from before, but now we see the distribution of points in the game time dimension. Since we had more data sets for shorter game times, this separation is logical. But, the points that are all by themselves tend to be slightly less accurate due to lack of data. Meaning that our classifier might be less confident due to different changes in the input data.

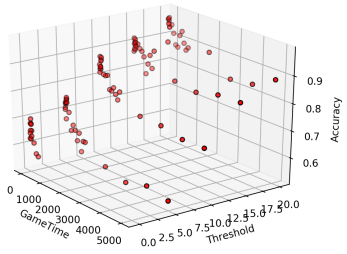


Figure 5: Relationship between all three variables: game count, game time, and accuracy from Python

After analyzing our prediction of the user. We decided to try and predict race and status. We added new decision trees for each attribute and then plotted all three in one graph. Still using the 3D graph, we can see in Figure 6, we were able to predict race the best, and status the worst. The status is just a win or loss. This is not an easy thing to predict only on key strokes. We would need other data points to support the keystrokes for a better prediction. The decision tree was definitely over fitting for the status attribute. On the contrary, it was surprising to see that we could better predict the race over the user. Seeing them side by side, our classifier worked best when the prediction didn't have too many options for what it could choose. Since there was only three races, the tree was too greedy when making classifications. In fact, we believe that the better prediction had to do with the data itself. Since each race relied on certain moves and abilities for which each player would be consistent in executing. Meaning the keys they pressed would be more the same for all players of that race. When predicting the player, every player will have a more unique key stroke pattern making the data more inconsistent and harder to predict.

Finally, we plotted one more graph Figure 7, using the tree depth of each decision tree made for all files and attributes. We wanted to compare accuracy with how big the tree became. And we can see that the best accuracy came from a smaller tree. This gave us important insight on how decisions work. When the tree is very large the complexity grows. Meaning the trees input is stable and doesn't vary greatly. Over fitting is another issue here where the tree doesn't do a a very good job of generalizing the data. This is directly correlated with what the classifier is trying to predict. Like we said earlier each attribute we tried to predict had an accuracy that made sense with the input data: players keystrokes. Overall, this graph showed us how good our input data was for creating a prediction. Having the tree depth be not too high or not too low meant a better prediction overall.

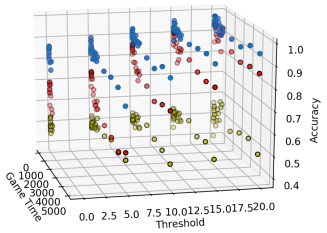


Figure 6: Relationship from Figure 5, Legend- **Blue**: Race, **Red**: User, **Yellow**: Status

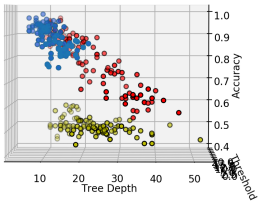
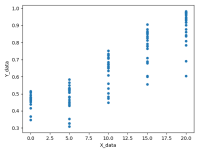


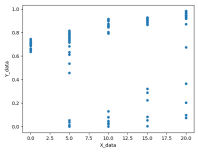
Figure 7: Relationship of accuracy to Tree Depth, Legend- **Blue**: Race, **Red**: User, **Yellow**: Status

Other classifiers

We also tried other classifiers to train our data and see the results. It is very easy to transfer from decision tree to another by Python. We just needed to change the model type and can get results. Figure 8a is what we got by SVC classifiers. Figure 8b is what we got from MLP classifiers.



(a) SVC



(b) MLP

Figure 8: Results of other classifiers

We can see that results we get from decision tree were better. And the performance of MLP classifier was not very stable Especially when game count threshold increases. This might caused by the number of data was not enough.

Conclusions

Overall, using python turned out to be easier in the end. Since we started in Knime, we were able to understanding how the classifier used a set and a target to create the tree. Though the nodes in Knime made things easy. Once we had plotted our graph in Knime we moved over to Python to create our graphs. Not only were we able to recreate our graphs. We were able to plot 3D graphs and use other classifiers. Using Python gave us a deeper understanding of the decision tree by having to manually split the data and create our own tables to then be plotted. We mostly focused on the decision tree classifier but we were able to interpret the data much better after seeing our output graphs. It was clear that some things are much harder to predict than others. An insight to decision trees that seemed like it would be of fault to the algorithm. But in our examples it was the data itself that made it difficult for the decision tree to be accurate. It would take a lot more data to be able to have a higher percentage accuracy for predicting a user. It might require multiple data sets over years to learn the keystroke fingerprint of each player. In conclusion, we were able to predict accurately the user and especially the race with very little data preparation.