CS420 - Artificial Intelligence

Lab 02 - Decision Tree

Nguyen Ngoc Bang Tam - 19125033

Table of contents

I. Overview

The structure inside the source folder (**`\src`**):

```
\ decision-tree
  \ decision-tree-xx.pdf
  \ decision-tree-80-depth-xx.pdf
  \ matrix-xx.jpg
\ main.py
\ dataset.py
\ input.py
\ tree.py
\ env.py
\ connect-4.data
\ requirements.txt
```

To run the program, install all the packages in `requirements.txt` and run the file `main.py`. **Notice:** `graphviz` package must be installed on the computer for the program to work. (for MacOS: `brew install graphviz`, for Linux: `sudo apt install graphviz`).

The folder **`decision-tree`** contains visualization and confusion matrices that are pre-run.

`connect-4.data` is the CSV-formatted dataset that is uncompressed from the `connect-4.data.Z` downloaded from the source. Each line in this file corresponds to a training example, including 43 comma-separated values where the first 42 values represent the features and the last value represents the label.

The entry point of the solution is `main.py`.

- In this file, the dataset in the above CSV is imported and shuffled (using the `read_input` function from `input.py`). Each of the feature values is mapped (from ‘b’, ‘x’, ‘o’ to 0, 1, 2) to be used in the Decision Tree model.

- Afterwards, 4 different proportions of train and test data are iterated. At each proportion, we split the dataset (using `split_train_test` in `input.py`), then create a `DecisionTreeWrapper` (imported from `tree.py`) that automatically fits a sklearn’s DecisionTree model on the input training data. It is followed by the visualization of the tree into a PDF file `decision-tree-xx.pdf`, and the evaluation of the tree using `classification_report` and `confusion_matrix`. The report is printed, while the matrix is saved to `matrix-xx.jpg`.

- Finally, we choose the 80/20 dataset to run the DecisionTree model with `max_depth` ranging from 2 to 7. (`max_depth=None` is the default case that is run above) At each depth, we draw the decision tree and save to `decision-tree-80-depth-xx.pdf`.

The following customizations can be made inside `env.py`:

- `INPUT_DATA`: File path to the input CSV file.
- `TRAIN_SPLITS`: List of proportion of train data.
- `TREE_VISUALIZE_PATH`: File path prefix of the PDF file of decision trees.
- `PLOT_TREE:` Turn on/off plotting the decision trees in the first section.
- `CHOSEN_PROP_INDEX`: The index of the proportion of train dataset (in `TRAIN_SPLITS`) chosen in the final part of examining the max_depth.
- `DEPTHS:` List of values for max_depth to experiment.

II. Evaluation of decision trees

After training the DecisionTree model using each of the train and test data subsets, we evaluate them using the functions `classification_report` and `confusion_matrix`.

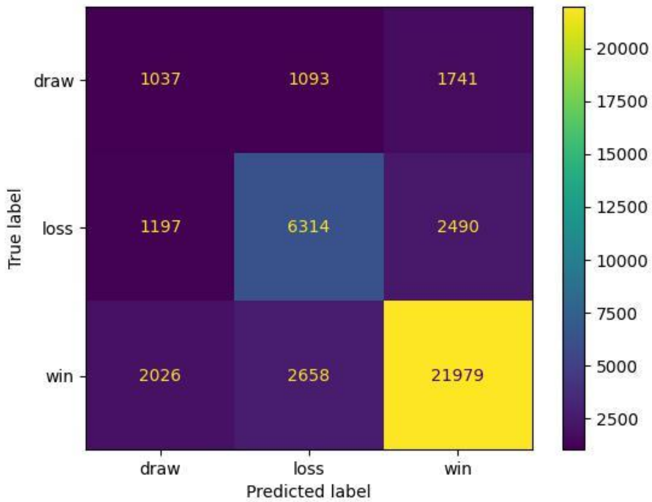
1. Train 40% - Test 60%

```
%% Train 40% - Test 60%
Train: 27022 - Expected train: 27022 - Test: 40535  - Expected test: 40535
Fitting...
Evaluating...
              precision    recall  f1-score   support

        draw       0.24      0.27      0.26      3871
        loss       0.63      0.63      0.63     10001
         win       0.84      0.82      0.83     26663

    accuracy                           0.72     40535
   macro avg       0.57      0.57      0.57     40535
weighted avg       0.73      0.72      0.73     40535
```

This model has the precision of 24% for draw, 63% for loss and 84% for win. This means that among the actual draws, losses and wins, the model can detect 24% of draws, 63% of losses and 84% of wins, while the rest are mistaken for another result. It can be deduced that this model is best at recognizing a win, but is not as good at recognizing a loss, and is very bad at recognizing a draw.

This model has the recall of 27% for draw, 63% for loss and 82% for win. This means that among all examples that are predicted to be draws, only 27% of them are actually a draw. The same thing happens with losses and wins. In other words, most of the model's predictions for a win are correct, while most of its predictions for a draw are wrong.

The precision and recall of this model for each label is relatively the same, both suggesting that the model is good at recognizing and predicting a win, but behaves poorly in recognizing a loss, or even worse when it comes to a draw. Also, since the precision and recall of each label is the same, the F1 score is also somewhere in between. The average accuracy of this model is 72%, but we know that mostly it can only predict a win precisely.

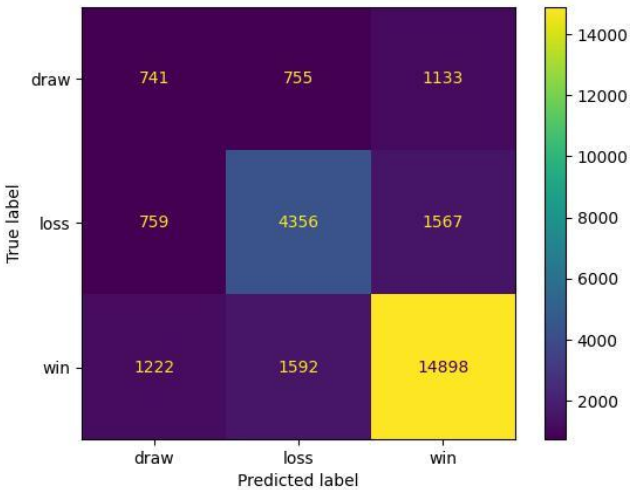


The confusion matrix also agrees with the deduction above, since the number of correct predictions for a win (21979) is high compared to the actual number of wins (26663), while the number of correct predictions for a draw (1037) is only one third of the actual draws (3871).

2. Train 60% - Test 40%

```
%% Train 60% - Test 40%
Train: 40534 - Expected train: 40534 - Test: 27023  - Expected test: 27023
Fitting...
Evaluating...
              precision    recall  f1-score   support

        draw       0.27      0.28      0.28      2629
        loss       0.65      0.65      0.65      6682
         win       0.85      0.84      0.84     17712

    accuracy                           0.74     27023
   macro avg       0.59      0.59      0.59     27023
weighted avg       0.74      0.74      0.74     27023
```

In this model, all metrics slightly increase, with the F1-score of draw from 26% to 28%, of loss from 63% to 65%, and of win from 83% to 84%. The average accuracy also increases slightly from 72% to 74%. However, we could still notice that the metrics for the ‘win’ label dominate significantly over the remaining labels.

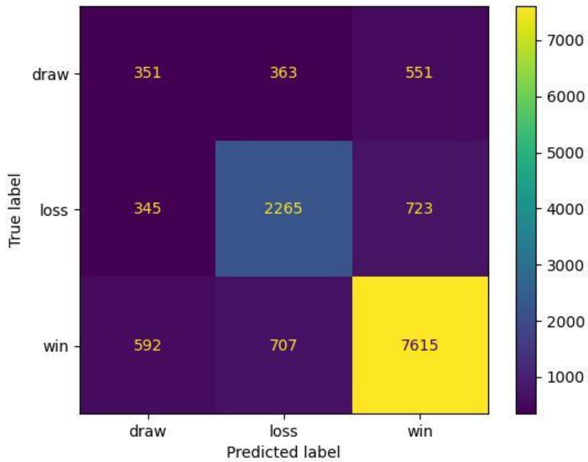


Since in this model, the number of examples in the test set is reduced, the number of predictions also decreases. However, the proportion of correct predictions agrees with the metrics in the above table.

3. Train 80% - Test 20%

```
%% Train 80% - Test 20%
Train: 54045 - Expected train: 54045 - Test: 13512  - Expected test: 13512
Fitting...
Evaluating...
              precision    recall  f1-score   support

        draw       0.27      0.28      0.27      1265
        loss       0.68      0.68      0.68      3333
         win       0.86      0.85      0.86      8914

    accuracy                           0.76     13512
   macro avg       0.60      0.60      0.60     13512
weighted avg       0.76      0.76      0.76     13512
```

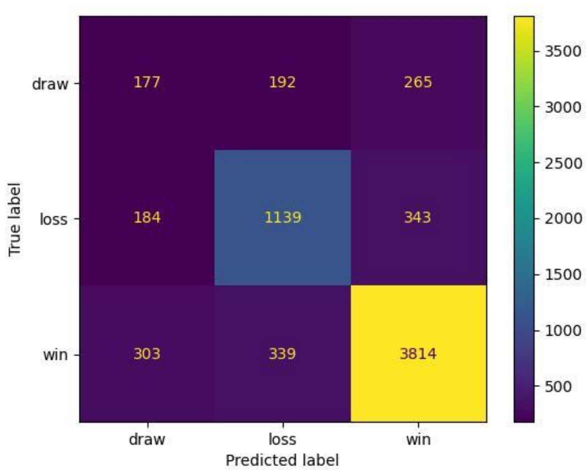
In this model, we see a small increase in the F1-score of loss (65% to 68%) and win (84% to 86%), while the metric for draw stays the same. The average accuracy increases slightly from 84% to 86%. This is by far the best model in terms of average accuracy as well as individual accuracies of each label. However, it still can only perform well in predicting wins, acceptably predicting losses, and behaves badly at predicting draws.



4. **Train 90% - Test 10%**

```
--------------------
%% Train 90% - Test 10%
Train: 60801 - Expected train: 60801 - Test: 6756  - Expected test: 6756
Fitting...
Evaluating...
              precision    recall  f1-score   support

        draw       0.27      0.28      0.27       634
        loss       0.68      0.68      0.68      1666
         win       0.86      0.86      0.86      4456

    accuracy                           0.76      6756
   macro avg       0.60      0.61      0.60      6756
weighted avg       0.76      0.76      0.76      6756
```

With yet another increase in the training set, however, we do not see any changes to the metrics of the prediction on the test data.



5. **Comments**

- As we increase the proportion of the train dataset, the accuracy of the decision tree model increases slightly, and so does the accuracy of predicting each individual label. However, as we reach a threshold (e.g: 80% for train dataset), the accuracy can no longer increase. Though not demonstrated in this report, the accuracy can even decrease due to the overfitting of the model.
- Since the number of examples labelled with 'win' dominates the 2 remaining labels, the model can only perform well in predicting this label while missing the knowledge to recognize draws and losses.
- The above metrics can differ in different runs of the program, due to the random shuffling of the dataset. However, the average result would still report the same meaning.

III. Evaluation of different max_depth

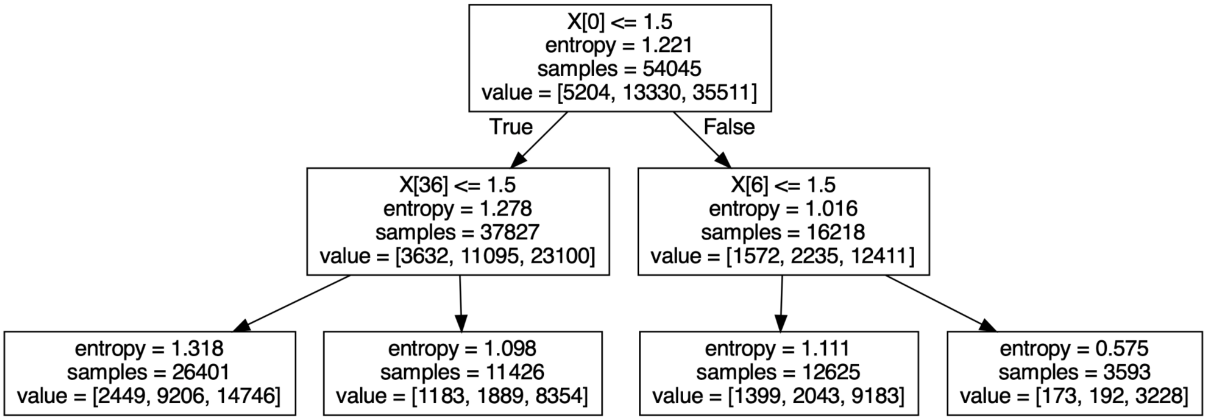
We experiment training the decision tree model on the 80/20 dataset with the constraint `max_depth` ranging from 2 to 7. We do not need to rerun the case where `max_depth` is None because it is exactly the model we ran in the previous section for the 80/20 dataset. The actual depth of this model is also printed:

```
%%%% max_depth = None: Actual depth = 39
%%%% max_depth = 2 running...
%%%% max_depth = 3 running...
%%%% max_depth = 4 running...
%%%% max_depth = 5 running...
%%%% max_depth = 6 running...
%%%% max_depth = 7 running...
```

The actual depth varies in each running of the program, indicating that if we do not limit the decision tree's depth, the tree can use up to 42 features to classify an example.

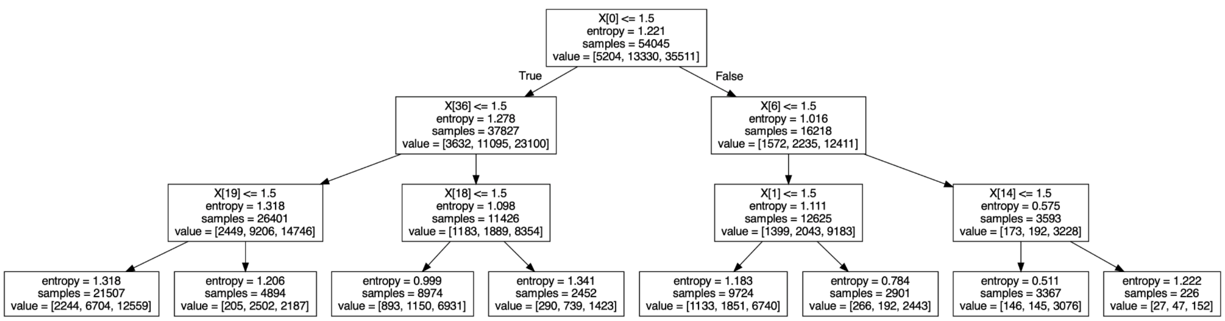
1. **Visualization**

At each depth, a visualization of the tree is saved to `decision-tree-80-depth-xx.pdf`. Some of the examples are shown below.



`max_depth=2`

Since we limit the depth to be maximally 2, we can only classify an example using 2 features: X[0] and either X[36] or X[6].



`max_depth=3`

On extending the maximal depth of the tree to be 3, we observe the appearance of one more feature in the process of classification: either X[19], X[18], X[1] or X[14].

The visualization of other depths from 4 to 7 can be found in the corresponding PDF file. For the case of None, it is represented in the PDF `decision-tree-80.pdf`.

It is worth noticing that the time taken to plot the decision trees of `max_depth` from 2 to 7 is in the matter of seconds, while the time to plot in the case of `max_depth=None` is significantly longer, in the matter of minutes. Hence, we always plot the decision trees in this section, regardless of the `PLOT_TREE` variable in `env.py`.

2. Accuracy

The accuracy of each model is shown in the table below:

Max_depth	Accuracy
None	0.76
2	0.66
3	0.67
4	0.68
5	0.69
6	0.69
7	0.70

As observed, as we increase the `max_depth` from 2 to 7, the accuracy of the decision tree gradually increases accordingly, from 66%, 67%, to 68%, 69%, 70%. This is because the larger the depth, the more features are used in classifying an example. It reaches a peak when the `max_depth` is set to None, which means no constraint is set to depth of the tree and the model is free in choosing any number of features (up to 42 features).