TDT4300: Data Warehousing and Data Mining

Assignment 4

Jørgen Wennberg Pettersen & Scott Jennings

jorgenwp@ntnu.no & sajennin@ntnu.no

April 10, 2025

1 KNN

1.1 Theoretical question

First we can start off by calculating all distances from the Green point to the Red and Blue points:

Blue Points:

- (4, 6): $min(|3-4|, |4-6|) = min(1, 2) = 1$
- (5, 5): $min(|3-5|, |4-5|) = min(2, 1) = 1$
- (7, 9): $min(|3-7|, |4-9|) = min(4, 5) = 4$
- (9, 3): $min(|3-9|, |4-3|) = min(6, 1) = 1$

Red Points:

- (0, 7): $min(|3-0|, |4-7|) = min(3, 3) = 3$
- (1, 6): $min(|3-1|, |4-6|) = min(2, 2) = 2$
- (6, 8): $min(|3-6|, |4-8|) = min(3, 4) = 3$
- (8, 2): $min(|3-8|, |4-2|) = min(5, 2) = 2$
- (9, 8): $min(|3-9|, |4-8|) = min(6, 4) = 4$

We can sort these by distance in ascending order:

Distance	Point	Class
1	(4, 6)	Blue
1	(5, 5)	Blue
1	(9, 3)	Blue
2	(1, 6)	Red
2	(8, 2)	Red
3	(0, 7)	Red
3	(6, 8)	Red
4	(7, 9)	Blue
4	(9, 8)	Red

Now we can identify the class for the green point using the k-NN classifiers with k = 1, 3, and 5 using majority voting with weighting. The weighing used here will be *inverse distance weighting*: $Weight = \frac{1}{Distance}$. This is to give closer points more influence.

- k = 1
  - Nearest: (4, 6) – Blue
- k = 3
  - (4, 6) – Blue, d=1 → w=1.0
  - (5, 5) – Blue, d=1 → w=1.0
  - (9, 3) – Blue, d=1 → w=1.0
- k= 5
  - (4, 6) – Blue, d=1 → w=1.0
  - (5, 5) – Blue, d=1 → w=1.0
  - (9, 3) – Blue, d=1 → w=1.0
  - (1, 6) – Red, d=2 → w=0.5
  - (8, 2) – Red, d=2 → w=0.5

This gives us the following classification results:

k	Predicted Class
1	Blue
3	Blue
5	Blue

1.2 Implementation

The implementation process consisted of three parts:

The dataset splitting was implemented to randomly shuffle sample indecision. This is to ensure unbiased splits. There was a small problem encountered with the distribution of the dataset, where the assertions where not passed. This was because of the use of integer type conversion and was fixed by utilizing the round python method instead to round the float to the closest integer.

The k-NN algorithm implementation has three main components; (1) distance calculation, (2) neighbour finding, and (3) classification. The distance calculation uses Euclidean distance to calculate the similarity between points. The neighbour detection simply finds the k closest training examples to a test point. The classification then uses majority vote to determine the final prediction.

Lastly, the evaluation simply calculates classification accuracy by iterating over the predictions and comparing them to the ground truth. This measures the proportion of correctly classified samples.

1.3 Conceptual question

On one side, accuracy is intuitive and straightforward to communicate. It's a natural way to understand the overall correctness of a classification model. And when evaluated on a test set of unseen data, accuracy provides an unbiased estimate of how well the model generalizes to new mushrooms [1].

However, accuracy can be misleading with imbalanced datasets. As the the course syllabus [1] explicitly state, accuracy may not be well-suited for evaluating models derived from imbalanced data sets. To understand this we can consider a hypothetical scenario with the mushroom dataset simalar to the one in the book. Suppose that edible mushrooms are far more common than poisonous ones (an imbalanced dataset). A classifier that always predicts mushrroms to be "edible" might achieve very high accuracy simply because it correctly

classifies the majority of instances. However, such a classifier is in reality completely useless for a company, as it would fail to identify any poisonous mushrooms, leading to potentially severe consequences.

Alternative metrics could provide a much more nuanced evaluation. To consider situations where class imbalance is a concern, alternative metrics such as precision, recall, and the F-measure are more appropriate. A confusion matrix would also offer more detailed information about the types of errors made by a classifier (false positives and false negatives).

2 Decision Tree

2.1 Implementation

For this task the dataset splitting method and evaluation method was the same as implemented in task 1. The implementation specific to the decision tree was simplified by using the scikit-learn python library. The `DecisionTreeClassifier` class was used for creating a decision tree classifier that was trained on the data. To select the best split, the model was configured to utilize Gini (degree of impurity) as the criterion. It's built in prediction method was then later used to predict the label for any given set of features. The trained model produced the following decision tree:

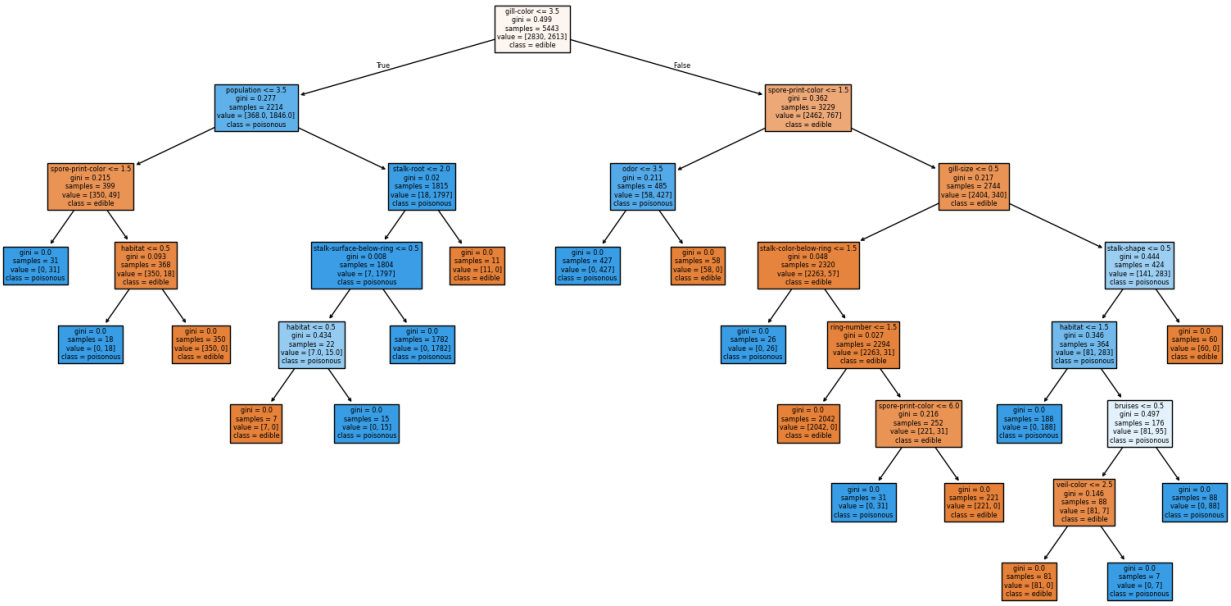

Figure 1: Decicion Tree produced by training the model

2.2 Conceptual question

a. The performance of the Decision Tree is better compared to the k-NN classifier. The Decision Tree model was created within less than a second and performed its predictions instantly. The k-NN classifier however, while not requiring training, needed 3 minutes and 44 seconds to complete all predictions. We can see that the Decision Tree also achieve a accuracy of 1, meaning that it's predictions are perfectly correct for the test set. Meanwhile, the k-NN classifier achieved a accuracy of 0.9992, showing that while it is close to perfect, there are still some errors in it's predictions.

b. When it comes to misclassified items, the Decision Tree outperforms the k-NN classifier by having 0 errors in it's prediction. This is compared to the k-NN's 2 misclassified items.

c. Considering the task at hand where misclassifying a poisonous mushroom as edible has severe consequences, recall for the "poisonous" class is a crucial metric. In other words, we want to minimize false negatives (failing to identify poisonous mushrooms). Having this in mind the preferred classifier for the company would probably be the Decision Tree. This is mainly because of its advantages when it comes to interpretability [1]. Understanding the reasoning behind the classification would likely be important to the company (e.g., to identify key characteristics of poisonous mushrooms). This is not something provided by k-NN classification. KNN doesn't explicitly learn a model, which might make it harder to understand which attributes are most indicative of poisonous mushrooms. Furthermore, if the "poisonous" class is rare, a new poisonous mushroom might not have many poisonous neighbours, leading to misclassification. For these reasons a decision tree might be preferred.

References

[1] Tan, Pang-Ning. Introduction to Data Mining, Second Edition. Pearson, 2018. Chapter 7.