Assignment 1 – Classification using Scikit-learn

Student Name: Gavin Skehan | Student ID: 21440824 | Programme: 4BCT

Algorithm 1: Decision Tree Classifier

A decision tree is a supervised learning model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The tree is constructed by recursively splitting the dataset into smaller subsets based on feature values, using if-then-else rules to approximate the outcome. Each internal node represents a feature, each branch represents a decision (rule), and each leaf node represents the outcome.

How the Algorithm Works

Using the Scikit-Learn build-in library we can configure the decision tree. The algorithm recursively splits the dataset based on feature values until each subset belongs to a single class, resulting in pure leaf nodes. The splitting process uses entropy to measure the uncertainty in each node, maximizing the information gain at each step.

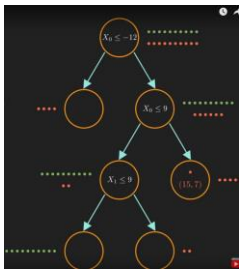


Figure 1: Decision Tree

Entropy is the measure of information contained in a state.

Entropy = – (P1) log(P1) – (P2) log (P2)

Where p1 and p2 are the percentage split of the classes in the node. **Information Gain** is the difference between the entropy of the parent node and the entropy of the child nodes.

Why I Chose Decision Trees

Decision Trees are easily interpreted and can handle both numerical and categorical data, making them suitable for the wildfires dataset. However, due to the number of features, there is a risk of overfitting, meaning the model may memorize patterns from the training data rather than generalizing.

Hyperparameter Tuning

- **max_depth:** Controls the maximum depth of the tree to prevent overfitting.
- **min_samples_split:** Ensures multiple samples inform every decision in the tree, controlling its growth.l

Model Training and Evaluation

Starting by defining both the training.csv and test.csv files and defining the independent columns and the dependent columns where 'Fire' is the target variable. Next, we map the target variable to binary values (1 for 'Fire' and 0 for 'No Fire). The decision tree was configured by using Scikit-learns default parameters. The default parameter for max_depth='none' and min_samples_split='2'. We train the model on these default parameters on the training set before evaluating the results from predicting the output on the training and test set.

Here are the accuracy scores for the default hyperparameters calculated by utilizing the metrics package:

```
Accuracy of default parameters on training set:  1.0
Accuracy of default parameters on test set:  0.86
```

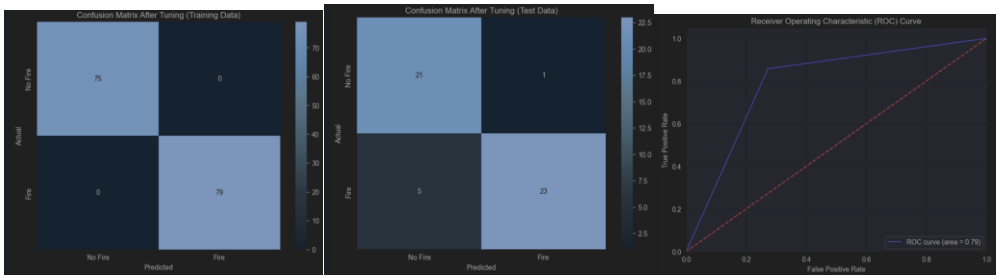
When training our model on the training set with default parameters, we achieved a perfect accuracy score of 1.00, this is a sign of overfitting meaning the model has become too accustomed to the patterns of the training data and may perform very poorly when given a new set of data it hasn't seen before. The model has an accuracy score on the test data of 0.86 which is good but room to improve this score through hyperparameter tuning.

Results After Fine-Tuning Hyperparameters

To find the most optimal solution for the hyperparameters, we created a loop to incrementally check every combination of max_depth and min_samples_split.

- Max_depth: we are checking depths of 1-32.
- Min_samples_split: we are checking (0.1, 1.0, 10)

From our results, we can derive that the best-performing combination of the two hyperparameters has a constant value of 15 for min_samples_split and can range from a max_depth of 3 to 32. This indicates that an increase in the depth of the decision tree doesn't correlate to better performance. This could hint at signs of saturation or overfitting.



ROC Curve: shows how well the model balances between detecting fires (True Positive Rate) and avoiding false judgments (False Positive Rate). An AUC of 79% indicates that the model correctly distinguished between fire and no-fire instances in 79% of cases, which is better than random guessing.

Accuracy Results: We achieved a test accuracy score of 88% (a slight increase) from 86%. We found these results by creating a loop to record the performance of each hyperparameter in H1 with each hyperparameter combination in H2 essentially creating two matrices.

Classification Report

Training Data: the classification report for the training set shows an accuracy score of 1.00, this is a perfect score meaning all predictions that were made are correct, and there are no false positives or false negatives. The precision, recall, and f1-scores are also 1.00 highlighting how well the model performed on the training set, although this is a sign of the model overfitting.

Test Data The classification report for the test data shows an accuracy score of 0.88, meaning the model predicted 88% of unseen data correctly. Both classes (0 and 1) had F1-Scores of 0.88 which means there is good harmony between precision and recall. This indicates a good performing model performance. For class 0 the precision was 0.81 and the recall was 0.95 whereas for class 1 the precision was 0.96 and the recall was 0.82. For class 0, the model is better at identifying class 0 instances but sometimes incorrectly labels class 1 instances as class 0. For class 1, the model is more conservative in predicting class 1 but it misses more actual class 1 instances than class 0.

Classification Report (training set after tuning):

	precision	recall	f1-score	support
0	1.00	1.00	1.00	75
1	1.00	1.00	1.00	79
accuracy			1.00	154
macro avg	1.00	1.00	1.00	154
weighted avg	1.00	1.00	1.00	154

Classification Report (Test set after tuning):

	precision	recall	f1-score	support
0	0.81	0.95	0.88	22
1	0.96	0.82	0.88	28
accuracy			0.88	50
macro avg	0.88	0.89	0.88	50
weighted avg	0.89	0.88	0.88	50

Algorithm 2: Logistic regression

Logistic Regression is a statistical method used for binary classification. It predicts the probability of an event (Fire: yes/no) given one or more features. We set thresholds based on our needs for the model to classify whether there will be a fire or not. The logistic regression model maps the output to a sigmoid function which produces an S-shaped curve. It assumes a linear relationship between the independent variables (features) and the log odds of the dependent variable (target).

How Logistic Regression Works

The model calculates the probability of the target class using the formula, we can access this algorithm through Scikit-Learn libraries:

P(class = 1) = 1 / (1 + e^(-(β0 + β1x1 + β2x2 + … + βnx_n)))

Where:

- P(class = 1) is the probability of the instance belonging to class 1 (Fire).
- Xn are independent variables that may influence the outcome (temperature).
- β0, …., βn are the regression coefficients, reflecting the strength and direction of the relationship between each feature and the outcome.

Sigmoid function: a weighted combination of features and applies the function to produce a probability estimate between 0 and 1. **F(x) = 1 / 1 + e^-x**

The model estimates the best-fitting coefficients using maximum likelihood estimation (MLE) to minimize the difference between the predicted and actual outcomes.

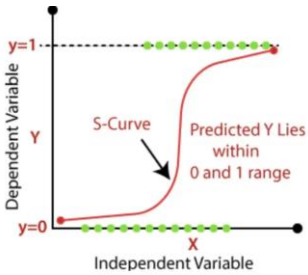


Figure 2: Logistic Regression (Sigmoid) Function

Why I chose Logistic Regression

Logistic Regression serves as a simple model and acts as a good baseline for our classification model. The complexity of our dataset features and their interactions may cause performance issues, so tuning may improve results.

Hyperparameter Tuning

Hyperparameter 1: Regularization Strength C

Controls the inverse regularization strength, with smaller values specifying stronger regularization. Lower values can prevent overfitting, whereas higher values increase the risk of overfitting but are more flexible. Tuning will allow us to get the best performance by balancing overfitting and underfitting.

Hyperparameter 2: Solver

A solver is an optimization algorithm used to minimize the cost function. Identifying the solver that suits our dataset will optimize the model's performance.

Reasons for selecting these hyperparameters

'lbfgs' performs well and is memory efficient but may have convergence issues. 'Sag' is faster for large databases while 'liblinear' is recommended when you have a high-dimension dataset. I will avoid using 'newton-cg' and 'saga' as they are computationally expensive. I will also be fine-tuning C (regularization strength) which must be a positive float. Smaller values specify stronger regularization and high value tells the models to give high weight to the training data.

Model Training and Evaluation

Starting by defining both the training.csv and test.csv files and defining the independent columns and the dependent columns where 'Fire' is the target variable. Next, we map the target variable to binary values (1 for 'Fire' and 0 for 'No Fire). The decision tree was configured by using Scikit-learns default parameters. The default parameter for solvers is lbfgs and the default parameter for C is 1.0.

Here are the accuracy scores for the default hyperparameters calculated by utilizing the metrics package:

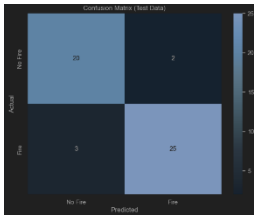
```
Accuracy of default parameters on training set:
 0.8701298701298701
Accuracy of default parameters on test set:  0.8
```

We can see after training our model on the training set provided, it achieved an accuracy score of 0.88%, and on the test data, 0.8%. There is no sign of overfitting in the logistic regression model but the test score leaves room for tuning.

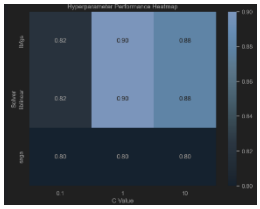
Results After Tuning Hyperparameters

To find the best hyperparameters, I created a loop to check each solver with each C value. You can visualize it as two matrices [a, b, c] and [x, z, y] where a, b, and c will map to each value in the second matrix. Once this process is finished the code identifies the most optimal, combination which was C=1 and solvers=lbfgs or liblinear. Both achieved an accuracy score of 90% increasing 12.5% from the default hyperparameters of 80%. This significant improvement transitions the model from good to excellent performance. This increase is created by tuning the model's performance and solving the convergence issue by adding the max_iter value.

```
Best performing hyperparameters:
C=1, solver=lbfgs
C=1, solver=liblinear
Best Test Accuracy: 0.9000
```



The confusion matrix identifies the accuracy of the model concerning the number of True Positives, True Negatives, False Positives, and False Negatives. The model performed very well with only 2 False Negatives and 3 False Positives.



The heatmap is used to highlight the accuracy scores of the model with the combinations of the solvers and the C values. There are nine outputs all displayed within the heatmap.

Classification Report

Training Data

The accuracy score on the training data is 0.87%, meaning the model predicted 87% of the training data classes correctly. As seen in the previous model class 0 has a higher recall of 0.92 compared to the lower precision score of 0.83. Class 1 has a higher precision score of 0.92 compared to the lower recall of 0.82. For class 0, the model is better at identifying class 0, it sometimes incorrectly labels class 1 instances as class 0. For class 1, it misses more actual class 1 instances than class 0. The F1-Score is 0.87 for both classes, there is a good balance between precision and recall.

Test Data

The model out-performs the accuracy on the training data with a score of 0.90% on the test data. This suggests the model generalizes well to unseen data, avoiding overfitting which is a sign when the training accuracy is much higher than the test accuracy. The precision and recall scores are very high with 0.87 and 0.91 for class 0 and 0.93 and 0.89 for class 1. The F1-Score for class

0=0.89 and class 1 is 0.91 indicating a very well-performing model. The slightly higher F1 score for class 1 hints at the model being more confident in predicting class 1 than class 0.

Classification Report (Training set after tuning):

	precision	recall	f1-score	support
0	0.83	0.92	0.87	75
1	0.92	0.82	0.87	79
accuracy			0.87	154
macro avg	0.87	0.87	0.87	154
weighted avg	0.87	0.87	0.87	154

Classification Report (Test set after tuning):

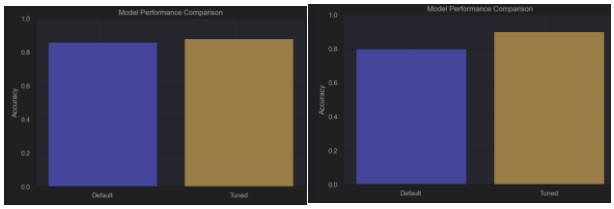
	precision	recall	f1-score	support
0	0.87	0.91	0.89	22
1	0.93	0.89	0.91	28
accuracy			0.90	50
macro avg	0.90	0.90	0.90	50
weighted avg	0.90	0.90	0.90	50

Comparative Analysis of Models

Both models performed well. Decision trees are prone to overfitting as we seen because of their ability to learn complex patterns and non-linear relationships. Logistic Regression generalizes better but it assumes a linear relationship between the features and the log-odds of the target, this can limit its capacity to capture the non-linear relationships.

Decision Trees can handle complex, non-linear relationships better than logistic regression. Accuracy was close with decision trees achieving a score of 88% and logistic regression, 90%. Decision tree accuracy went from 86% - 88% and logistic regression went from 80% - 90%.

Visual Comparison: Performance increase From Tuning: Logistic Regression / Decision Tree



Conclusion

Key Findings: Hyperparameter tuning increased the performance of both models. Logistic Regression saw a big jump following the tuning while the decision tree saw a slight increase. We could tell there was an overfitting issue when training the decision tree. The logistic regression model was able to generalize the data better. The tuning has a greater impact on logical regression but as we saw in the decision tree the depth doesn't seem to make much of a difference past a value of 3 although min_samples_split of 15 was the most idea value leading to improved performance.

Model Performance: Logistic regression performed better on the test data after the hyperparameter training. The decision tree achieved an accuracy of 88% compared to the logistic regression accuracy score of 90% after tuning. Logistic regression improved by 12.5% from tuning while the decision tree improved by 2.3256%. The result is that logistic regression performed better once tuned.

Suggestions for further improvements: We could explore other ensemble methods like random forests as the number of trees will help remove overfitting as seen in decision trees. It would also be good to use a validation set and standardize the data for logistic regression as this will prevent overfitting. For logistic regression, we could explore polynomial logistic regression to better handle non-linear relationships.

References

1. Scikit-learn. (2007 – 2024) 'Supervised Learning', ' 1.1.11 Logical Regression' / '1.10 Decision Trees', 30/10/24. Available at: (https://scikit-learn.org/stable/supervised_learning.html)

2. Courseteach. (2023) 'Supervised learning with scikit-learn (Part 3)-Classification using Scikit-Learn', August 12, 2023. Available at: (https://medium.com/@Coursesteach/guide-to-supervised-learning-with-scikit-learn-part-3-c31b01c547f9)

3. Nerd, Normalized. (2021). 'Decision Tree Classification Clearly Explained!', 13 Jan, 2021. Available at: (https://www.youtube.com/watch?v=ZVR2Way4nwQ&ab_channel=NormalizedNerd)

4. (France, Francesco. (2023), 'Logistic Regression Algorithm, Jul 20, 2023. Available at: https://medium.com/@francescofranco_39234/logistic-regression-algorithm-6451c7928375)

5. Figure 1: 'Nerd, Normalized. (2021). 'Decision Tree Classification Clearly Explained!', 13 Jan, 2021. Available at: (https://www.youtube.com/watch?v=ZVR2Way4nwQ&ab_channel=NormalizedNerd'

6. Figure 2: '(France, Francesco. (2023), 'Logistic Regression Algorithm, Jul 20, 2023. Available at: https://medium.com/@francescofranco_39234/logistic-regression-algorithm-6451c7928375)'

7. Ben Fraj, Mohtadi, (2017), 'InDepth: Parameter tuning for Decision Tree' , Medium, 20/12. Available at: (https://medium.com/@mohtedibf/indepth-parameter-tuning-for-decision-tree-6753118a03c3)

8. Leung, Kayli. (2019), 'How to Code and Evaluate of Decision Trees', Medium, 16/11. Available at: (https://medium.com/swlh/how-to-code-and-evaluate-of-decision-trees-2d94093b3c1a)

9. Glavin, Frank, 2024, 'Code and Dataset Examples for Topic Three', Canvas. Available on canvas.

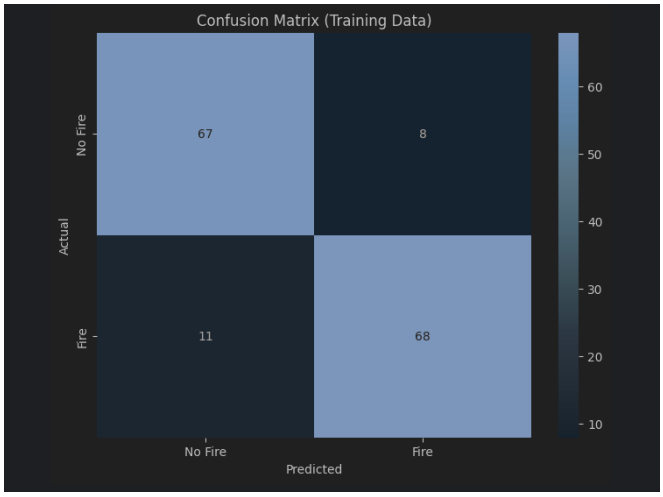
Appendix

I experimented with whether to standardize the initial model for logistic regression (pre-processing is not necessary for this assignment) when training on the training set. I concluded that I should standardize the data even though the accuracy of the non-standardized model was slightly better. The reasoning is due to a convergence error. Logistic regression relies on the gradient descent method to find the best parameters, when features are on very different scales the optimization path can become inefficient leading to the warning. The small improvement could be misleading due to the convergence issue therefore it is much more important to standardize the data to remove the convergence problem. The True Positives and False Positives are the same which is also a key consideration.

Standardizing the data

Classification Report (Training set):Classification Report (Training set):

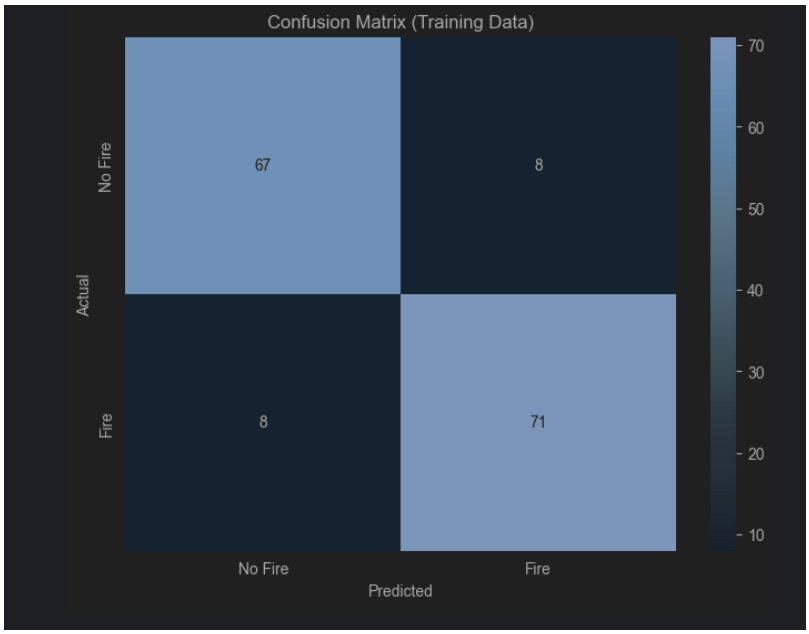
	precision	recall	f1-score	support
0	0.86	0.89	0.88	75
1	0.89	0.86	0.88	79
accuracy			0.88	154
macro avg	0.88	0.88	0.88	154
weighted avg	0.88	0.88	0.88	154



Non-StandardizedData

Classification Report (Training set):

	precision	recall	f1-score	support
0	0.89	0.89	0.89	75
1	0.90	0.90	0.90	79
accuracy			0.90	154
macro avg	0.90	0.90	0.90	154
weighted avg	0.90	0.90	0.90	154



Validation Sets

It‘s a good idea to create validation sets when fine-tuning models. Validation sets are generally made up of 20% of the training data. It acts as test data so you don’t have to expose your model to the test data until the most optimal model is chosen. This can prevent overfitting as the model is tested on new data instead of training data.