Name: Andrew Hayes
Student ID: 21321503
Programme: 4BCT

CT4101

2024–10–19

Assignment 1: Classification Using Scikit-Learn

1 Description of Algorithms

1.1 Algorithm 1: Random Forest

Random decision forest is a supervised machine learning algorithm that can be used for both classification & regression that builds upon the **decision tree** algorithm by combining several decision trees to generate labels for a dataset. An implementation of this algorithm for classification is provided in scikit-learn as `sklearn.ensemble.RandomForestClassifier`[3]. While it can be used for regression as well as classification, I will only be referring to its use as a classification algorithm in this assignment, as regression is not relevant to the wildfire classification task at hand.

Since the random decision forest algorithm builds upon the decision tree algorithm, it is first necessary to explain briefly what decision trees are and how they work. A decision tree can be thought of a series of internal nodes (i.e., nodes which are not leaf nodes) that contain a question which separates the input data. The decision tree is traversed from root to leaf for each instance being classified, where the leaf node to which we arrive is the label for that instance. For example, a decision tree might be used to determine whether or not a living thing is a mammal, where each internal node is a question that helps to separate non-mammalian data instances from mammalian, and each leaf node is a label stating whether or not the living thing is a mammal. Each internal node should narrow down the final label as much as possible i.e., each question should give us the maximum information about the instance and should be arranged in the order that narrows it down as quickly as possible.

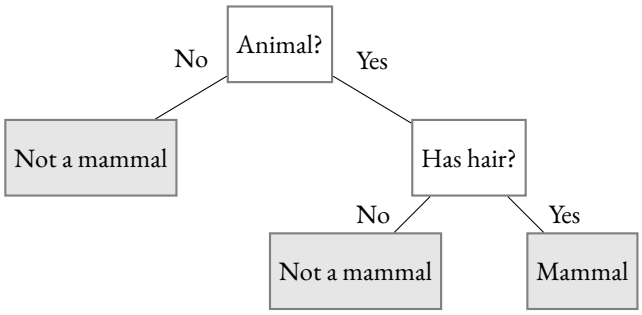

Figure 1: Simplified Decision Tree to Determine Whether a Creature is a Mammal

Decision trees have many advantages: they are visualisable by humans and aren't "black-box", they can model non-linear relationships easily, and they are robust to outliers. However, they have their disadvantages, including instability (small changes in the training data can significantly alter the tree structure) and in particular **overfitting**: when the algorithm fits too exactly to the training data, making it incapable of generalising to unseen data. An extreme example of overfitting would be if the example decision tree above started to ask far too specific questions, e.g. "Is it a dolphin", "Is it a human". While this would have excellent performance & accuracy on the test data, it would not work at all for an animal it hadn't encountered before.

Random forests work by combining many decision trees into a forest, thus improving accuracy & reducing overfitting by averaging multiple trees, reducing variance and leading to better generation. These decision trees are each generated using random, potentially overlapping subsets of the data training data. While the original random forest algorithm worked by taking the most popular label decided on by the set of trees[1], the scikit-learn `RandomForestClassifier` works by taking a probability estimate for each label from each tree and averaging these to find the best label[2].

In `RandomForestClassifier`, each tree is generated as follows:

1. A subset of the training data is randomly selected (hence the "Random" in the name of the algorithm). These subsets are selected "with replacement" which means that different trees can select the same samples: they are not removed from the pool once they are first selected. This results in unique, overlapping trees.
2. Starting with the root node, each node is *split* to partition the data. Instead of considering all features of the samples when choosing the split, a random subset of features is selected, promoting diversity across the trees. The optimal split is calculated using some metric such as Gini impurity or entropy to determine which split will provide the largest reduction in impurity.

3. This process is repeated at every node until no further splits can be made.

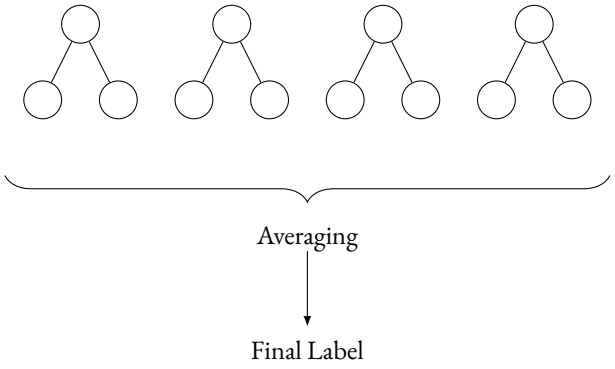

Figure 2: Random Forest Algorithm Example Diagram (with scikit-learn Averaging)

I chose the random forest classifier because it is resistant to overfitting, works well with complex & non-linear data like the wildfire data in question, handles both categorical & numerical features, and offers a wide variety of hyperparameters for tuning. It also has many benefits that are not particularly relevant to this assignment but are interesting nonetheless: it can handle both classification & regression tasks, can handle missing data, and can be parallelised for use with large datasets.

1.1.1 Hyperparameter 1: `n_estimators`

The hyperparameter `n_estimators` is an ***`int`*** with a default value of 100 which controls the number of decision trees (*estimators*) in the forest[3]. Increasing the number of trees in the forest typically improves the model's accuracy & stability, with diminishing marginal returns past a certain value, at the trade-off of increased computation & memory consumption. Each tree is independently trained, so there is a big trade-off between computational cost & performance. Using a lower number of estimators can result in underfitting, as there may not be a enough trees in the forest to capture the complexity of the data.

1.1.2 Hyperparameter 2: `max_depth`

The hyperparameter `max_depth` is an ***`int`*** with a default value of **`None`** which controls the maximum "depth" of each of the trees in the forest[3], where the "depth" of a tree refers to the longest path from the root node to a leaf node in said tree. With the default value of **`None`**, the trees will continue to grow until they cannot be split any further, meaning that each leaf node either only contains samples of the same class (i.e., in our case each leaf node is a definitive "yes" or "no") or contains a number of samples lower than the `min_samples_split` hyperparameter. The `min_samples_split` hyperparameter determines the minimum number of samples required to split a node; it has a default ***`int`*** value of 2 and therefore, since I am not tuning this hyperparameter for this assignment, it has no relevance as any leaf node that doesn't reach the minimum number of samples to be split is a "pure" node by virtue of containing only one class.

High `max_depth` values allow for the trees to capture more complex patterns in the data, but can overfit the data, leading to poor testing accuracy. Bigger trees also naturally incur higher computational costs, requiring both more computation to create and more memory to store. Lower `max_depth` values result in simpler trees which can only focus on the most important features & patterns in the data, which in turn can reduce overfitting; however, low values run the risk of creating trees which are not big enough to capture the complexity of the data, and can lead to underfitting.

1.2 Algorithm 2: C-Support Vector Classification

1.2.1 Hyperparameter 1: `kernel`

1.2.2 Hyperparameter 2: `C`

References

[1] Leo Breiman. "Random Forests". In: *Machine Learning* 45 (2001).

[2] scikit-learn Documentation. *Ensembles: Gradient boosting, random forests, bagging, voting, stacking*. URL: `https://scikit-learn.org/stable/modules/ensemble.html` Accessed on: 2024-10-06.

[3] scikit-learn Documentation. *`RandomForestClassifier` API Reference*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html` Accessed on: 2024-10-06.

[4] scikit-learn Documentation. *Support Vector Machines*. URL: https://scikit-learn.org/stable/modules/svm.html Accessed on: 2024-10-06.

[5] scikit-learn Documentation. *SVC API Reference*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC Accessed on: 2024-10-06.

[6] IBM. *What is random forest?* URL: https://www.ibm.com/topics/random-forest Accessed on: 2024-10-06.

[7] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.