Bourry Amir

Tree Predictors for Binary Classification

19/06/2024

Statistical Methods for Machine Learning

I. Introduction

This report presents the implementation of the DecisionTree model and the RandomForestClassifier for binary classification. The implementations were then tested on datasets and compared with scikit-learn.

Initially it was planned to test the implementations on the Mushroom dataset from ucimlrepo, but it turned out that the dataset was poorly indexed on some items, making its processing too complex even for scikit-learn. So to show the effectiveness of our Model we tested our implementation on the Breast Cancer dataset, again from ucimlrepo.

We are gonna define the two main notions of our project, the DecisionTree Model and the RandomForestClassifier.

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the datafeatures. A tree can be seen as a piecewise constant approximation. (definition from scikit-learn.org).

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Trees in the forest use the best split strategy, i.e. equivalent to passing splitter="best" to the underlying DecisionTreeRegressor. The sub-sample size is controlled with the “max_samples” parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree. (definition from scikit-learn.org).

II. Decision Tree

A) How it works

To create our DecisionTree model, we created a class called DecisionTree in Python, as well as the fit and predict methods, which are used to train the model and predict the data classes respectively.

To train a DecisionTree, the main idea is to recursively divide the data into subsets, choosing the best possible division at each stage. The best division is the one where, after the division, the subsets are as homogeneous as possible, i.e. the classes are as homogeneous as possible. This is generally measured using the Gini index or entropy. In our case, we will use the entropy calculated by the following formula:

$$\psi(p) = -p * \log_2(p)$$

The entropy will be calculated such as :

$$Entropy = \sum_{i=1}^{c} \psi(p_i)$$

With p_i is the proportion of data points belonging to class i and c is the number of classes.

In our implementation, the entropy is calculated from the following function :

```
def _entropy(class_probabilities: list) -> float:
    return sum([-p * np.log2(p) for p in class_probabilities if p > 0])
```

The entropy is a measure of the randomness or disorder within a set of data. It is used to quantify the impurity achieved by a potential split of the data. Since our goal in construction a decision tree is to split the data in a way that results in the most homogenous subsets possible, we will need to find to lowest entropy possible. This metric will give us a low entropy if the splits have a dominant class, in order words, the data within the subset is mostly made up of a single class, leading to low disorder. On the other hand, if the entropy is high, this will mean that the subset is subject to disorder, which is not what we want since it will mean that we are going to need more splits after the one executed.

The mathematical explanation of entropy is based on its formula. When a dataset is split, the entropy of each resulting subset is weighted by the proportion of instances in that subset relative to the original set. The goal is to achieve splits that results subsets with the lowest entropy possible. When a subset after a split has a dominant class, the value p_i for that class is high (close to 1), and the values for other classes are low (close to 0). This situation leads to low entropy because the log of a number close to 1 is close to 0 by definition of the function.

Now, when we evaluate two potential splits in a decision tree, we can determine which split is better by comparing their entropy scores. The lower entropy score indicates which split we must choose. For example, if we consider two features with numerical values. We can create splits by using the median values of feature 1 and 2. We then examine the entropies of the resulting groups. If the split of feature 1 produces a lower entropy compared to the split based on the feature 2, we will choose the feature 1 to execute the split of the node.

At each level of our decision tree, an algorithm is used to determine which feature has to be chose to execute a split. This method is referred to as greedy approach because it selects the best split at each step without considering future splits. As a result, while the greedy approach finds the locally optimal split at each level, it may not always produce the globally optimal decision tree.

B) Implementation

To implement a decision tree, we used a tree data structure, which is inherently recursive. This structure is composed of elements from the TreeNode class. Each instance contains several components. They facilitate the decision-making process and the organization of the tree. Each TreeNode has a left child and a right child (they can be another TreeNode or null if the Node is a leaf or has only one child). The two branches linking the parent node and the child nodes represents the two possible outcomes of a decision at that node.

Additionally, each TreeNode stores the feature being considered, for a numerical split it would be for example the threshold, for a categorical split it would be the category to choose. This information will determine how the data is divided at the node. We also store the relationship between the

children of the TreeNode, ensuring that the flow of decisions from the root to the leaves follow a coherent path.

The recursive nature of this data structure means that each node can itself be the root of a subtree, with its own left and right children. That allows the tree to grow and expand as necessary to fit the data. This is crucial for the scalability of decision trees since we want to build it from the root to its leaves.

Our code includes a function ("create_tree") that allows to recursively create a tree, but the problem is that we need to stop it a some moment, otherwise we will have a infinite tree with infinite number of splits.

```
def _create_tree(self, data: np.array, current_depth: int) -> TreeNode:
```

To ensure our decision tree does not grow indefinitely and remains efficient, we need to implement stop criteria. In our case, we have chosen three stop criteria :

The maximum depth of the tree, this will limit the number of levels in the tree, preventing it from becoming too complex and overfitting the data.

The minimum number of samples in a node after division, this ensures that nodes do not split if they result in subsets that are too small, which might not provide meaningful insights.

Minimum information gain, this prevents splits that do not significantly improve the predictive index of the tree, ensuring that each division adds at least a substantial value.

These are defined when the tree is created.

```
# Check if the max depth has been reached (stopping criteria)
if current_depth > self.max_depth:
    return None
```

```
# Check if the min_samples_leaf has been satisfied (stopping criteria)
if self.min_samples_leaf > split_1_data.shape[0] or self.min_samples_leaf > split_2_data.shape[0]:
    return node
# Check if the min_information_gain has been satisfied (stopping criteria)
elif information_gain < self.min_information_gain:
    return node
```

C) Training and Prediction

To train our model, we used the fit method, which takes the training data and associated labels of the data set as parameters. This function constructs the decision tree by learning from the training dataset. It allows the model to understand the underlying patterns and the relationship between the data elements. During the prediction phase, we estimate the probability that a given sample belongs to a class. Each leaf node in the three holds fixed probabilities for each label, which are determined during the training phase. These probabilities reflect the proportion of each class in the subset of the data that reaches that leaf.

To make a new prediction, we begin with the unlabeled data at the root node of the tree and follow the decision rules down the tree. At each node, the decision criteria allows to go to the next level of the tree. It filters the the data until we reach a leaf. This process can be implemented using a "while loop" that iterates through the nodes of the tree, stopping only when there are no more nodes to traverse. At this point, the class probabilities stored in the leaf node are used to make the finale prediction.

D) Performance

To evaluate the performance of our model, we conducted testing using the Breast Cancer dataset obtained from the ucimlrepo. This dataset served as a benchmark to assess the accuracy and effectiveness of our model in classifying breast cancer cases. We meticulously compared our model's performance against that of scikit-learn (a famous library known for its robust implementations).

We also evaluated our performance by testing with other datasets such as Iris and a different version of the Breast Cancer dataset sourced directly from Sklearn. All dataset will be linked at the end of this report.

When evaluating the Iris dataset and the Sklearn version of the Breast Cancer dataset, our model exhibited slightly lower accuracy rates compared to the scikit-learn implementations. For instance, while scikit-learn achieved a high accuracy of 99% on these datasets, our model achieved a (still commendable) accuracy of 96%. This minor difference

suggests that while our model performs well, it might get better with some adjustments.

Overall, these comparations validate the effectiveness of our model using different datasets.

III. Random Forest Classifier

Now that we have an implementation of DecisionTree, we can implement a RandomForest. In our implementation we will use the same principle as DecisionTree. A RandomForest class which will contains DecisionTree and which will train and predict them.

A) How it works

RandomForest operates by generating multiple decision trees trained on diverse subsets of the data. Each decision tree independently predicts the class for new data instances, and the final prediction is determined by aggregating the results from all trees using a majority voting scheme. This ensemble approach helps mitigate individual tree biases and variance, resulting in a more robust and reliable prediction model.

To create these diverse subsets of data, RandomForest utilizes the bootstrapping method. This involves generating multiple random samples from the original dataset, each sample being of size determined by a hyperparameter called bootstrap_sample_size. Typically, this parameter controls the size of each subset, ensuring variability and reducing the risk of overfitting. The number of these subsets (we got it from the hyperparameter n_base_learners) defines how many trees will be trained in the forest.

During training, each decision tree is built on one of these bootstrapped samples, ensuring that each tree sees a unique portion of the data. The diversity among the trees allow our RandomForest to capture different aspects of the data patterns and improve the predictive performance.

B) Implementation

To implement our RandomForest, we employed a list data structure to stores and manage the decisions trees. The fit method was utilized for training these decision trees on different subset of the data, while the

predict methode facilities the prediction of class labels for new data based on the vote of all trees.

```
class RandomForestClassifier():
    def __init__(self, n_trees=10, max_depth=5, min_samples_leaf=1, min_information_gain=0.0, \
                 numb_of_features_splitting=None, bootstrap_sample_size=None) -> None:
```

Hyperparameters were defined to restricts the behaviour of RandomForest model:

“n_trees” parameter specifies the number of decision trees to be created in the ensemble. When we increases this parameter, the model robustness is increased but we also increase the computational complexity because more trees are gonna be created and trained.

“numb_of_features_splitting” parameter determines the number of features randomly selected and considered for splitting each node of a decision tree. This helps to increase diversity among the trees and help prevent them from relying too much on a single feature.

“bootstrap_sample_size” sets the size of the boostrap sample used to train each decision tree. It helps control the diversity and independence of each tree’s training data subset.

“max_depth” defines the maximum depth of the decision tree that compose the RandomForest. Restricting tree depth helps prevent overfitting by ensuring that trees do not become overly complex to the training data.

“min_samples_leaf” specifies the minimum number of samples required in a leaf node after a split. It makes that each leaf contains sufficient data. This constraint ensures that each leaf node contains enough instances to make correct predictions, preventing the model from learning overly specific patterns that may not generalize well on predicting a label.

“min_information_gain” sets the threshold for the minimum information gain required to perform a split at a node. It helps to make that that splits made by the decision trees lead to meaningful reductions in uncertainty, enhancing the overall predictive power of RandomForest.

By choosing those hyperparameters, we aim to find the right balance between how complex our model is and how well it predicts.

C) Training and Prediction

To predict using our RandomForest model, we aggregate the predicted probabilities from each trained base learner. By averaging these probabilities across all classes, we determine the final predicted probability for the RandomForest model. This approach ensures that our RandomForest is robust in its prediction by combining the insights of multiple decision trees.

D) Performance

To evaluate how effective is our model, we tested using the Breast Cancer dataset sourced from ucimlrepo. We rigorously compared the performance of our model against sckikit-learn's implementation and found that both gave comparable results, which can affirm the reliability and the accuracy of our approach.

Furthermore, we entended our testing to include the Iris dataset and another version of Breast Cancer dataset from Sklearn (the same datasetshave been used to stay consistent in our testing approach than the ones for DecisionTree implementation and test). We observed that our model performed very well across these datasets, despite earlier tests showing slightly lower performance compared to the scikit-learn implementation of the decision trees. This shows that our RandomForest implementation is reliable and predicts well across different datasets.

IV : Statistical Analysis of the approach

A) The Tree Predictor

1) Entropy :

Entropy H(X) is a measure of the impurity and uncertainty in a dataset before it is split at a node. For a classification problem with K classes and probability p_i for class i :

$$H(X) = -\sum_{i=1}^{K} p_i \log_2(p_i)$$

p_i represents the proportion of samples in class i at a given node.

2) Information Gain (will be useful for RandomForest) :

Information gain is the reduction in entropy achieved by splitting a dataset at a particular node of the tree. It quantifies how much uncertainty is reduced after the split. Given a split of dataset X into subsets X_j with $|X_j|$ denoting the number of elements in subset X_j.

$$Gain(X, X_j) = H(X) - \sum_j \frac{|X_j|}{|X|} H(X_j)$$

Where :

$H(X)$ is the entropy of the parent node.

$H(X_j)$ is the entropy of subset H_j.

In practice, decision tree algorithms using entropy select the feature and split point that maximizes information gain at each node. The process is in 4 steps. The first one is the calculation of entropy, then it evaluates splits (for each feature, it calculates the information gain that would result after the split based on that feature), it then chooses the best split and do a recursive splitting.

B) The Random Forest Classifier

Entropy and Information Gain are the same than the ones used for tree classifiers so we won't rewrite them.

1) Random Forest Ensemble :
   a) Bootstrap Sampling :

Random Forest trains multiple decision trees using bootstrap samples from the original dataset. Let X be the original dataset of size N and X^b denote a boostrap sample of size N . Each decision tree T_b is trained on a different example.

$X^b = \{(x_i, y_i)\}\ \forall i \in [1, N]$ where $(x_i, y_i) \sim X$.

   b) Feature Randomness :

For each decision tree T_b, a random subset of features $F_b \subset F$ is selected, where F is the set of all features. The size of F_b is specified by the hyperparameter numb_of_features_splitting. This means that each decision tree in the Random Forest considers a different subsets of features. It helps increase the diversity among the trees.

c) Prediction :

During prediction, each decision tree T_b in the forest independently predicts the class probabilities $\widehat{p_b}(y|x)$ for a given input x . The final prediction $\hat{p}(y|x)$ for class y is typically determined by majority voting for classification tasks :

$$\hat{p}(y|x) = mode(\{\widehat{p_1}(y|x), \widehat{p_2}(y|x), \dots, \widehat{p_B}(y|x)\})$$

Where B is the number of trees in the Random Forest. The predicted class it the class label that occurs most frequently among the predictions of all decision trees $T_1, T_2, \dots, T_B$. This approach ensures that the final prediction reflects the collective decision of the ensemble.

V : Conclusion

In conclusion, we've explored the fundamentals of decision tree classifiers and Random Forests. Decision trees use entropy to measure uncertainty in data, guiding how they split and make predictions. Random Forests extend this by combining many decision trees, each trained on different data subsets and using random features. This ensemble approach improves accuracy and robustness by averaging predictions across all trees or using majority voting for final decisions.

Entropy helps decision trees decide the best splits based on information gain, reducing uncertainty as they branch down. Random Forests then use multiple decision trees, each providing its own prediction. By averaging these predictions or selecting the most common one (mode), Random Forests boost accuracy and handle diverse datasets well.

Overall decision trees and Random Forests are powerful models. It is a effective way to do binary classification on datasets and make reliable predictions. They balance simplicity (not hard to implement) with

predictive power (0.95+ of accuracy most of the time), making them useful and not cheap in terms of time and complexity for various real-world applications (such as Breast Cancer detection which we used in this project, or Weather prediction such as shown in the lectures of this class). They make decisions given features which is crucial since a label is attributed given many parameters.

VI : Bibliography

- The Breast Cancer Dataset from the UCI Machine Learning Repository : http://archive.ics.uci.edu/dataset/14/breast+cancer
- Iris Dataset from Scikit-Learn : https://scikit-learn.org/1.5/auto_examples/datasets/plot_iris_dataset.html
-

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Please note that an AI tool has been used in the writing of this report to correct English mistakes, as the author is not a native English speaker. I also certify that the AI was not used to obtain information, to explain or describe the components of the project or for any other purpose other than linguistic correction.

BOURRY Amir, Erasmus Student from France.