TASK

Decision Trees I

Visit our website

Introduction

WELCOME TO THE FIRST DECISION TREES TASK!

In this task, we describe tree-based methods for regression and classification. Tree-based methods solve problems using a flowchart-like structure that is simple and easy to interpret.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

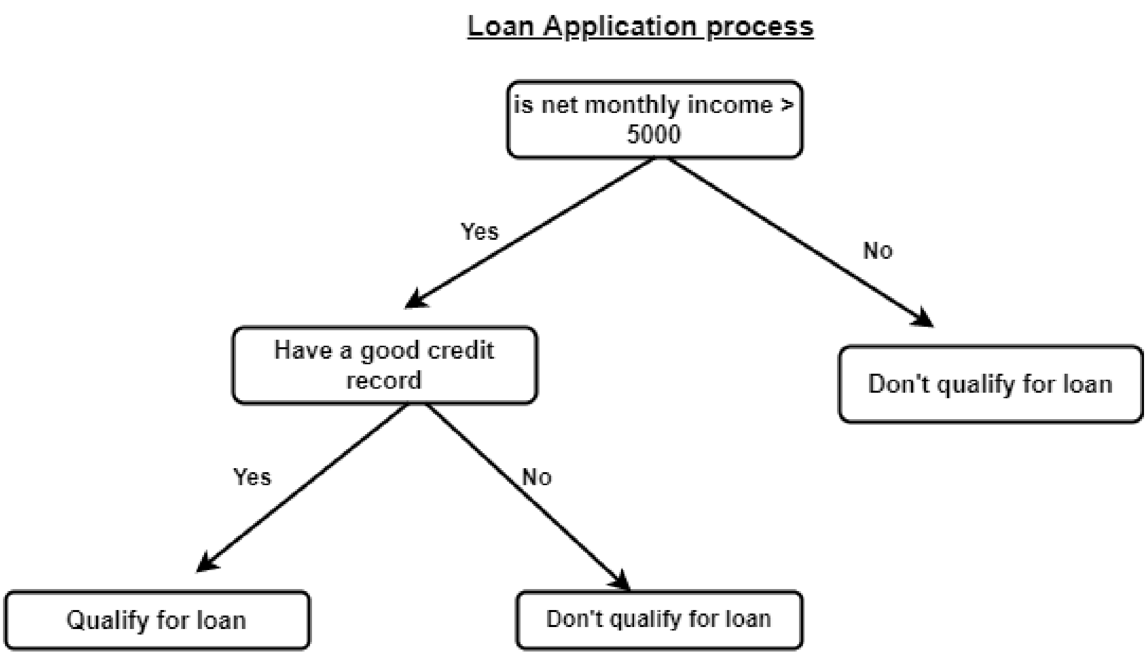
The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

INTRODUCTION TO DECISION TREES

Decision trees work by formulating simple rules that partition data into ever-smaller regions. Each partitioning is like a fork in the road, where a decision must be made. The decision is made based on *rules* which are derived from previous experiences. Decision trees are among the most interpretable machine learning techniques because they resemble the way humans make decisions.

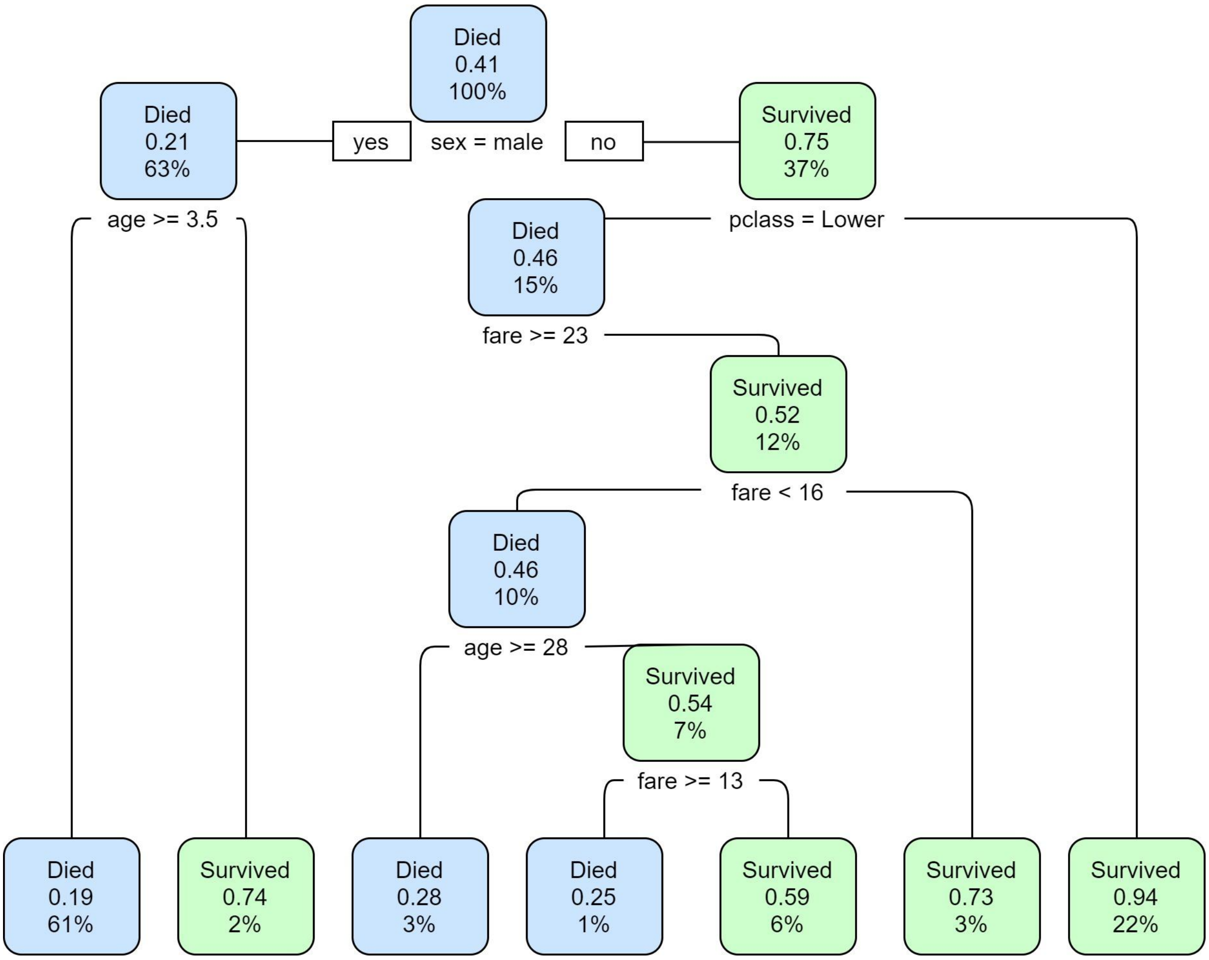
To use a toy example, if you need to figure out how to get dinner for the evening, you might first ask yourself whether there are enough ingredients in the fridge to make a meal. If the number of ingredients in the fridge is too low, you need to consider other options. Based on the time of day, you may decide to go to the shops for new ingredients or instead order a take-away. This decision-making process can be visualised in a tree-like diagram:



For all decision trees, you start at the top, follow the paths that describe the current condition, and keep doing that until you reach a decision. Note that the decision - the bold text in the diagram - is made at the "leaves" of the tree (the end of a branch, with no more branches coming off it).

Classification Trees

Decision trees created for datasets with a categorical dependent variable are called classification trees. As an example, let's look at the Titanic dataset. A tree model of this dataset shows us the likelihood of different kinds of passengers surviving the sinking of the Titanic. The tree consists of nodes, and each node has a rule that determines whether an instance moves on to the left or right child node. At the end of each possible path is a leaf. In a classification tree, a leaf contains the predicted label for an instance with those input features.

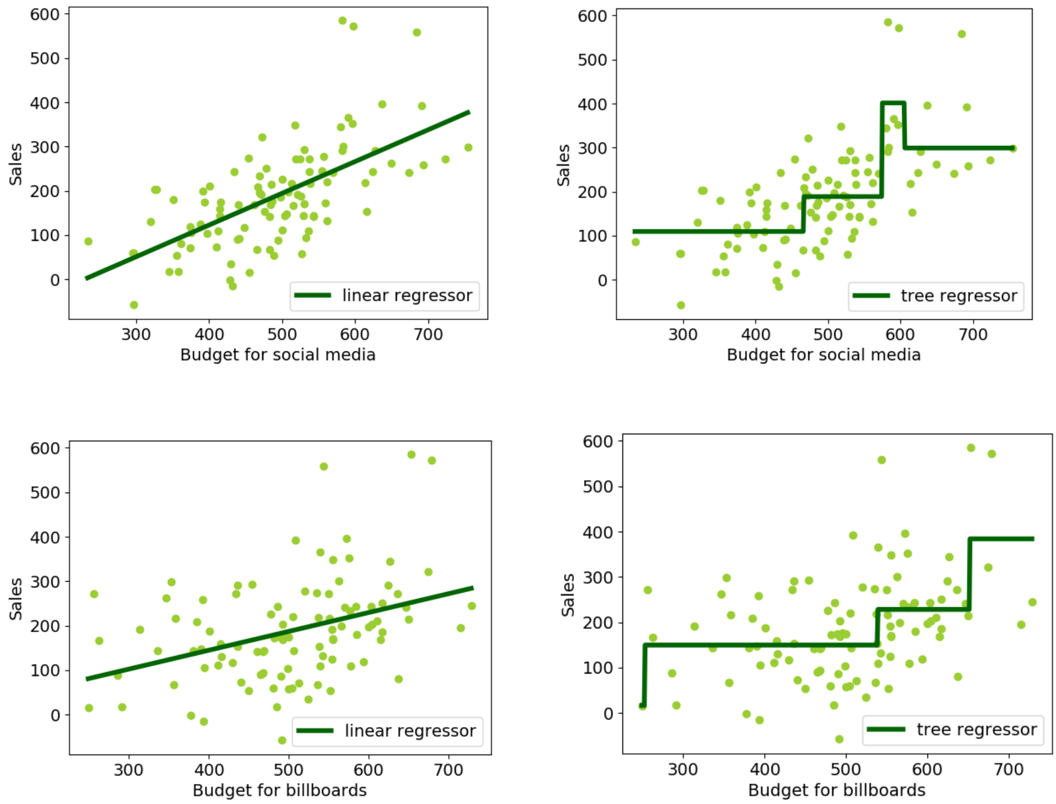


This particular tree also shows at each node what the probability of survival is. Without any prior knowledge, a passenger's chance of survival was 41%. But if we know that an instance is male, survival is a lot less likely. This tree shows that the lowest chance of survival was for adult males.

Regression Trees

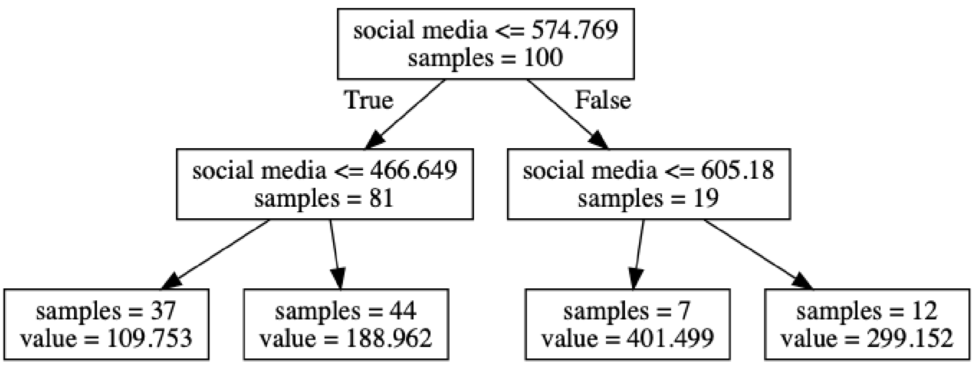
Decision trees can also be used for problems with a numerical dependent variable. A regression decision tree divides input features into regions and assigns categories to those regions. The regions and their labels are based on what best fits the data. Again, best fit here means the set of regions that minimises the distance between the predictions of the model and the observed values.

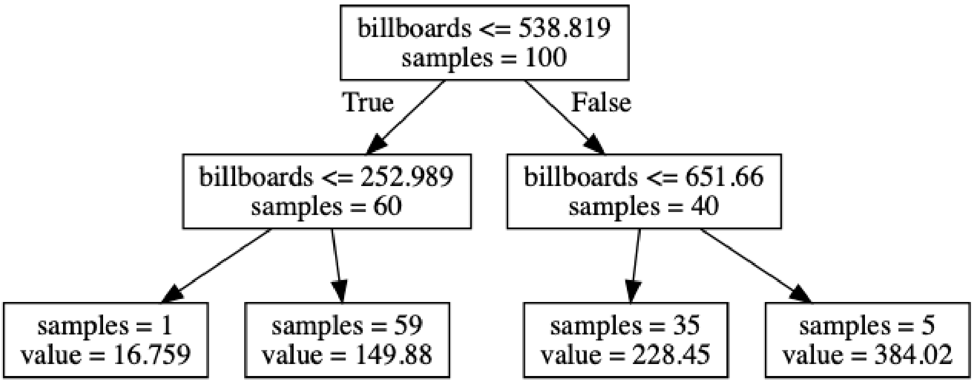
So while a linear regression approach to our advertising problem from Task 2 gave different predictions for each unique value of x, a regression tree gives different predictions for regions of x.



You can imagine that this approach is more flexible. The regions could capture a steeper or less steep increase in sales if that fits the data better than a linear model. The regions can also be arbitrarily specific or broad. Trees can also be visualised easily to get information about the decisions the model makes, in case we want to change the parameters of the model, as we will discuss in a bit.

Consider the following diagrams.





To interpret these diagrams, imagine a value for the social media budget of, say, 400. Since this value is below 574.769 and below 466.649, the model predicts that 109.753 items will be sold with this budget. This prediction is based on 37 samples in the data - you can follow the branches to the leaf node that shows this on the left-hand side of the above image.

OVERFITTING AND UNDERFITTING

When discussing trees it needs to be said that due to their flexibility, they are prone to something called "overfitting". Overfitting is one of the biggest causes for poor performance of machine learning algorithms, together with its counterpart, underfitting. Let's consider these concepts in more detail.

Underfitting

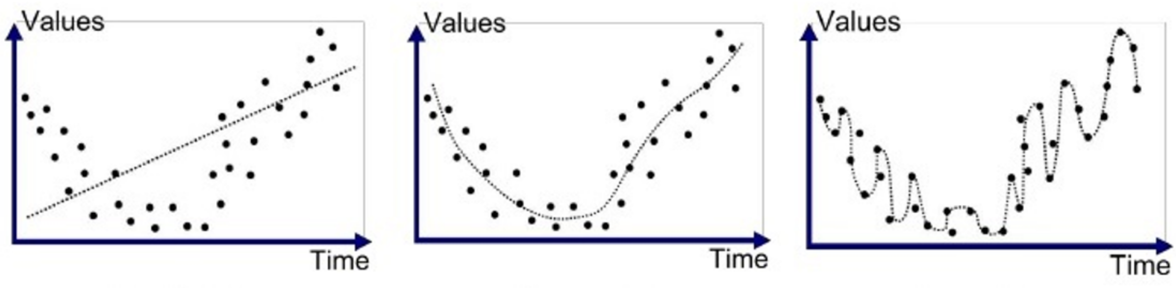
Underfitting refers to a model that can neither model the training data nor generalise to new data. The training error is high because the model was not able to make good predictions based on the input features. A model may fail to fit the data because it is too simple to capture the patterns, but underfitting is more commonly caused by a problem with the task set-up (e.g. the features *x* are not a good predictor of *y*) or with the training data (e.g. there is too little training data to learn from, or it contains too many mistakes).

Overfitting

An underfitting model has low training error, but will not perform well on test data. However, that does not mean that a model with low training error is automatically going to do well on test data. A model with very low training error may suffer from overfitting: some of the rules the model learned are only applicable to training data and are not useful for solving the overall problem we want to solve. The model is too tailored and does not generalise well. It is important that the model is able to generalise beyond the training data, as this data is only a sample. If the model is overfitted, it will perform very well on training data, but it won't translate to good performance on test data.

Diagnosing overfitting and underfitting

What are the causes of overfitting and underfitting? There are many reasons, and it can be typically difficult to determine. Generally speaking, an overly-complex model will overfit, and an overly-simple model will underfit. Think of overfitting as simply "remembering" the training data piece by piece, without learning how to construct a rule. Examine the three graphs below:



Underfitted Good Fit/Robust Overfitted

The first graph on the left is an incredibly simple model: it's just a straight line. There is no way that a straight line will be able to predict that data: hence you will see a high training error and a high testing error.

The middle graph is probably the best fit for the data. I'm sure that you can picture more samples of this data being added, and the model being able to give a rough estimate of what that should be. Note that the training error will be lower than an underfit, but higher than an overfit model. However, the testing error will be lower than both - testing error is what is most important. Let this be a lesson: zero error is rarely a good thing, and lower training error doesn't always mean a better model.

The last graph on the right shows a model that has overfit. The training error here will be incredibly low - practically zero! That being said, if more data were added, it's likely that the model will make a valuable prediction, as it doesn't seem to follow the overall trend of the data.

Fixing Overfitting and Underfitting

Theoretically, fixing overfitting and underfitting should be easy. If it is underfitting, then use a more complex model. If it is overfitting, then use a simpler model. However, this isn't always the case, and there could be a number of issues causing overfitting and underfitting. However, this is generally a good starting point.

When fixing overfitting, sometimes a simpler model will cause underfitting majorly. In some cases, we want to keep *some* of the complexity of our model, but not all of it. This can be fixed using a technique called **regularisation**. This is an important technique present in many (if not all) ML applications. In short, regularisation is the technique of penalising more complex components of the model during training, to force the model to minimise its usage. Let's say that you have a complex model that looks something like this:

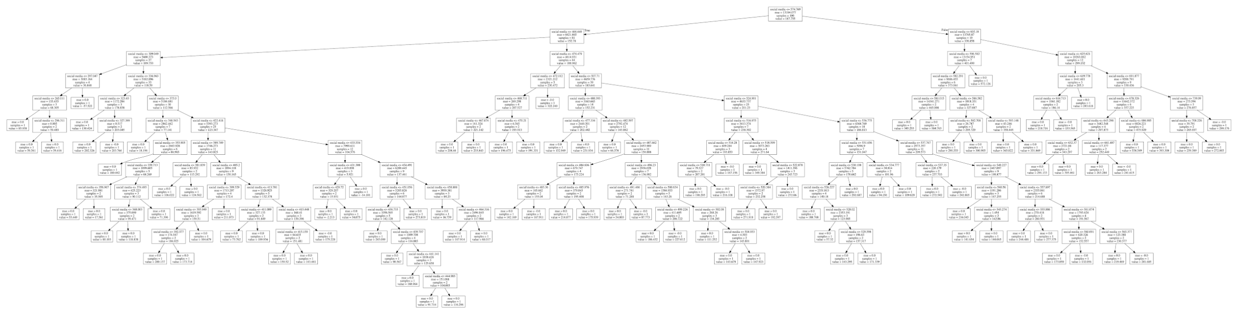
$$y = ax^5 + bx^4 + cx^3 + dx^2 + ex^1 + f$$

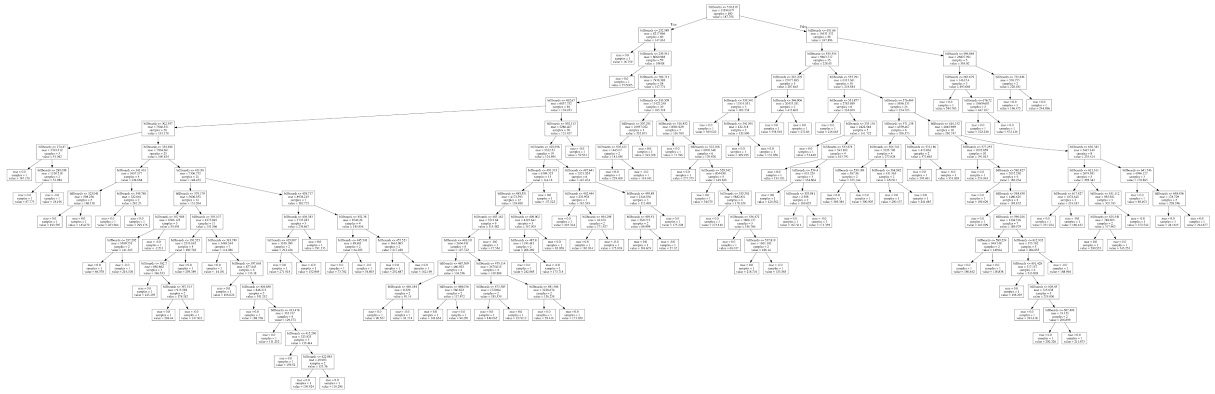
And you need to find the values of a, b, c, d, e, and f. This model is overfitting. However, you can't construct anything simpler than this, as it underfits the data. During training, you use the Mean Squared Error to calculate the loss function. However, you can also add something like $5a + 4b + 3c + 2d + e$ to the error. This means that the model will benefit more by using lower values of the higher-order parameters.

Pruning

As mentioned before, decision trees are very flexible and, therefore, are likely to overfit training data. This problem can be addressed by "pruning" a tree after training in order to remove some of the detail it has picked up and make the model more abstract and more general.

A strategy to prevent overfitting is thus to grow a very large tree without any restrictions first, which is then pruned to retain a more general subtree. In fact, that is how the example trees for the advertising budgets were created. If each of those trees were left to develop without restrictions, they would have become this detailed:





DEVELOPMENT DATA

How do we determine the best way to prune a tree? We can take subtrees of an unpruned tree and compute the test error of predictions of that subtree. We can then select the subtree with the lowest test error rate. However, once we have done that we have fitted our pruning parameter to the test data. This means the test data is no longer completely unseen. Fitting to test data can be avoided by splitting the dataset into three parts instead of two before training: a training set, **development** set ('dev set'), and test set. The development set is used to see whether the model seems to be generalising well to data that is not in the training set. This makes it possible to spot and try to remedy under- or overfitting. Only at the end do we test the model on totally unseen data. Another term for this third type of held-out data is the **validation** set.

Instructions

- See the **Decision_Trees.ipynb** that comes with this Task for an example of how to implement a Decision Tree.

Compulsory Task I

Follow these steps:

- Create a Decision Tree that can predict the survival of passengers on the Titanic. Make sure not to impose any restrictions on the depth of the tree.
- Load the **titanic.csv** dataset into a Jupyter notebook. This dataset comes from **here**.
- Select relevant variables from the data and split the data into a training, development, and test set.
- Train a decision tree and make a plot of it.
- Compute your model's accuracy on the development set.
- For tree pruning in Sklearn we usually use the `maxdepth` parameter, a parameter which determines how many levels the tree can have. Try building your model with different values of the max_depth [2-10]. At each step, create a plot of your tree and store the accuracies on both the training and development data.
- Plot a line of your training accuracies and another of your development accuracies in the same graph. Write down what shape the lines have and what this shape means.
- Pick an optimum value for the max_depth parameter and train your final decision tree using this parameter
- Report the accuracy of your final model on the test data.

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Things to look out for:

1. Make sure that you have installed and set up all programs correctly. You have set up **Dropbox** correctly if you are reading this, but **VS Code** or **Anaconda** may not be installed correctly.
2. If you are not using Windows, please ask a reviewer for alternative instructions.

Completed the task(s)?

Ask an expert to review your work!

Review work

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.