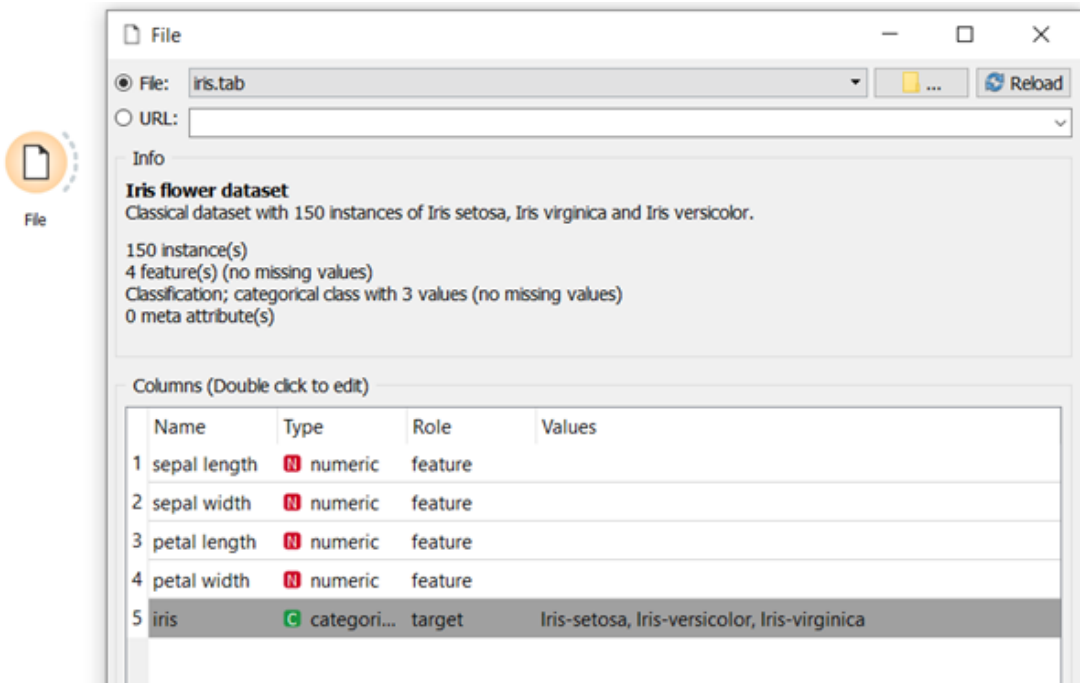
CSCU9M5 Practical 4

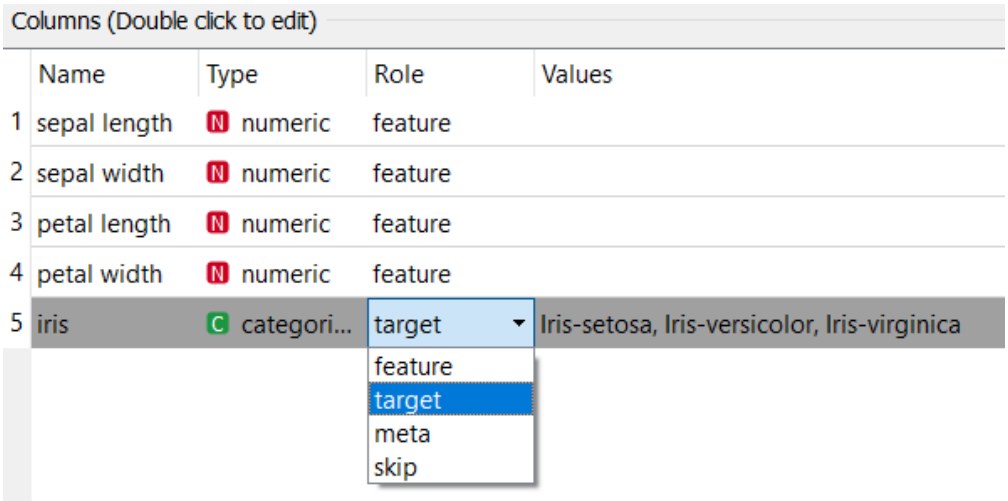
Training and Testing a k-NN model

Now we are going to train and test a k-NN classification algorithm using Orange.

Create a new workflow and add a file widget as you did previously, and double click it. You should see that the default file is iris.tab (if it is not, load in the iris.csv file for this session). This is the same dataset that was used in the videos for this session, and is a very well known dataset in machine learning.

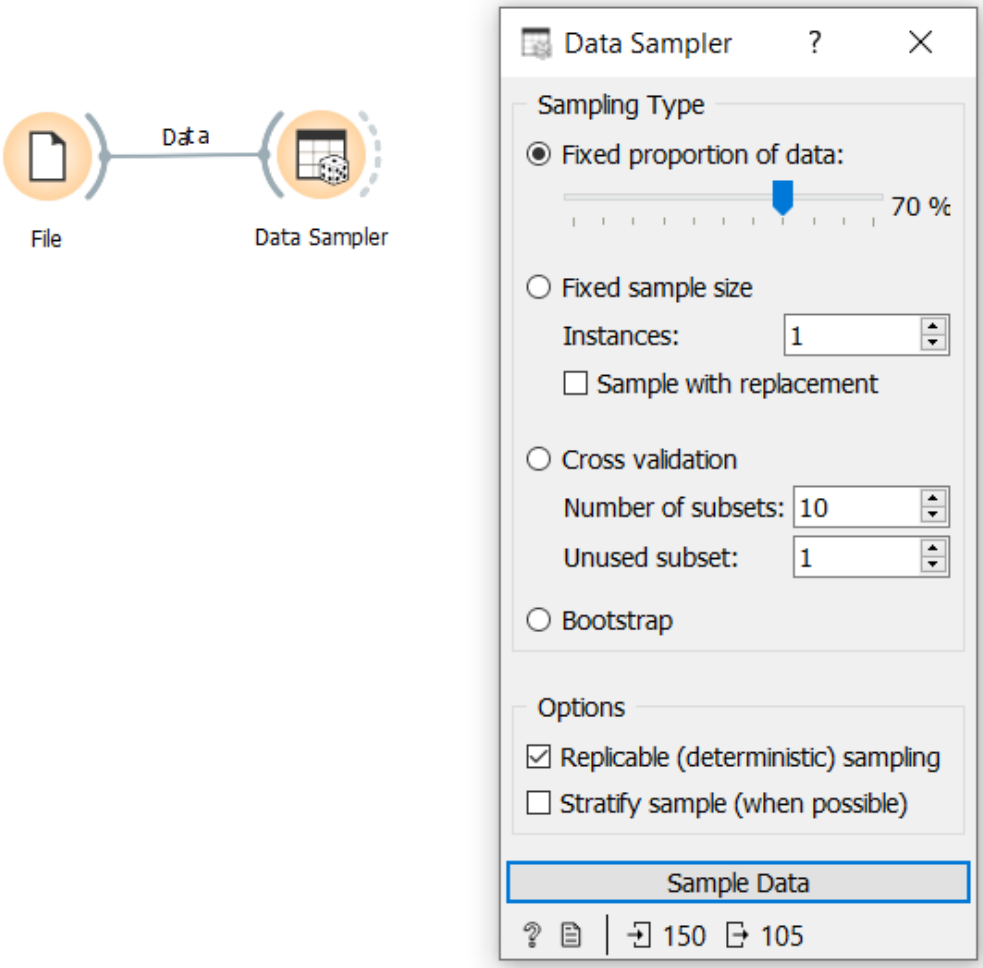


You will notice that the 'iris' variable role is set to 'target' - if it not is not set automatically in your instance, double click it and change it in the drop down, as seen in the image.

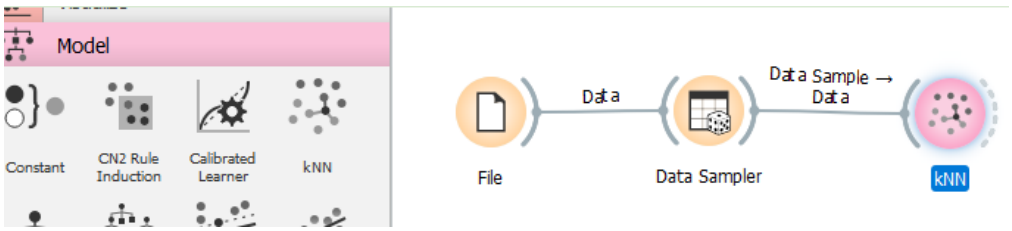


You will notice that the value column displays Iris-setosa, Iris-versicolor, and Iris-virginica - the three species of Iris that are labelled in this dataset.

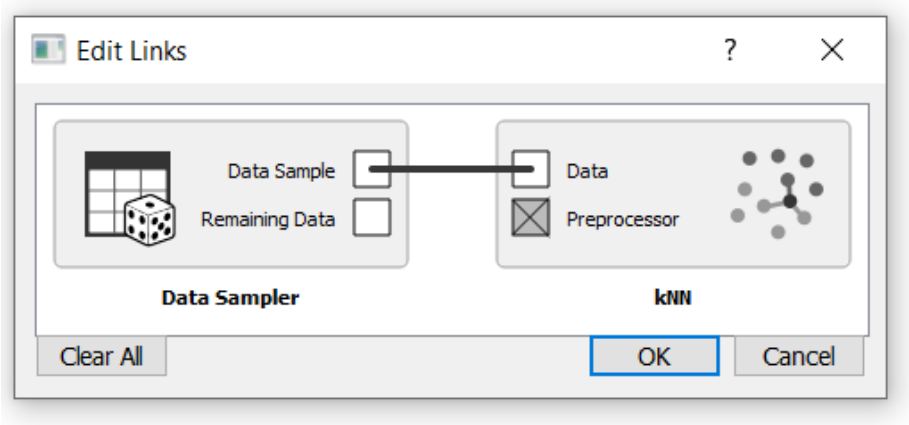
As we have done previously, we will split our dataset into Training and Testing data to evaluate our model using a Data Sampler. Connect a Data Sampler widget to the File widget, and double click it to ensure that the data is being split into two fixed proportions: 70%, which we will use to train the model, and 30%, which we will use to test it.



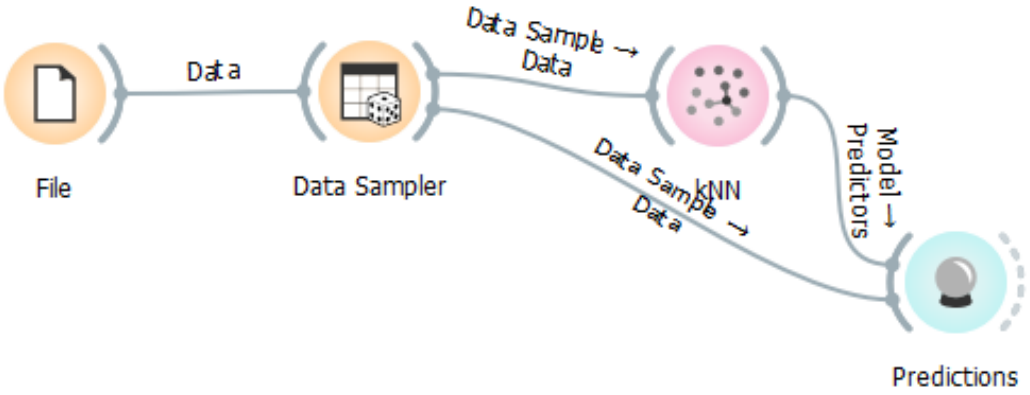
Now that we have our training data, it's time to create our kNN model. Add a kNN widget from under the Model section on the left of the screen, and connect the data sampler to it.

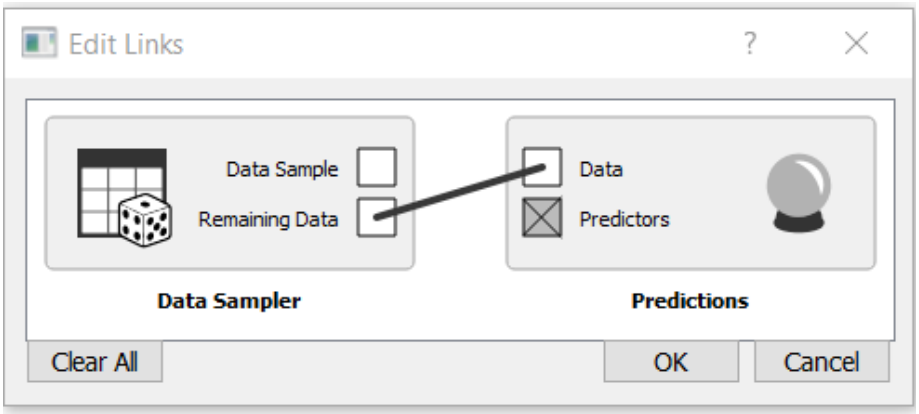


Double click the line joining the data sampler to the kNN widget, and ensure that the Data Sample from the Data Sampler is connected to the Data in the kNN. This means that the sample consisting of 70% of the overall dataset is now being provided to train the model.

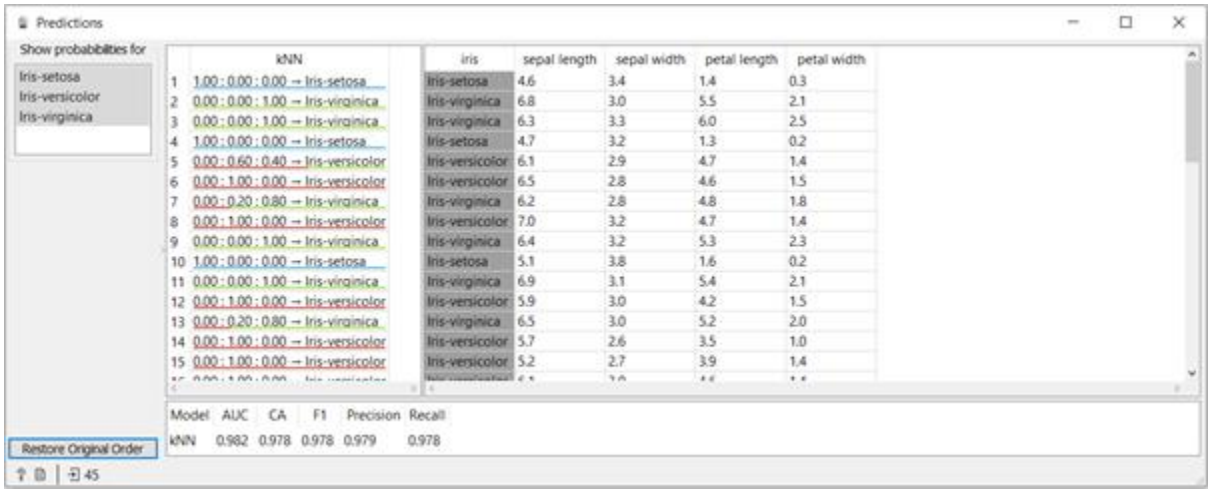


Now that we have a kNN model, we need to evaluate it. Do this by adding a predictions widget to the workflow. Connect the model, and connect the Data Sampler to this widget (ensure that the data going from the sampler to the prediction widget is the remaining 30% of the data as in the image below).





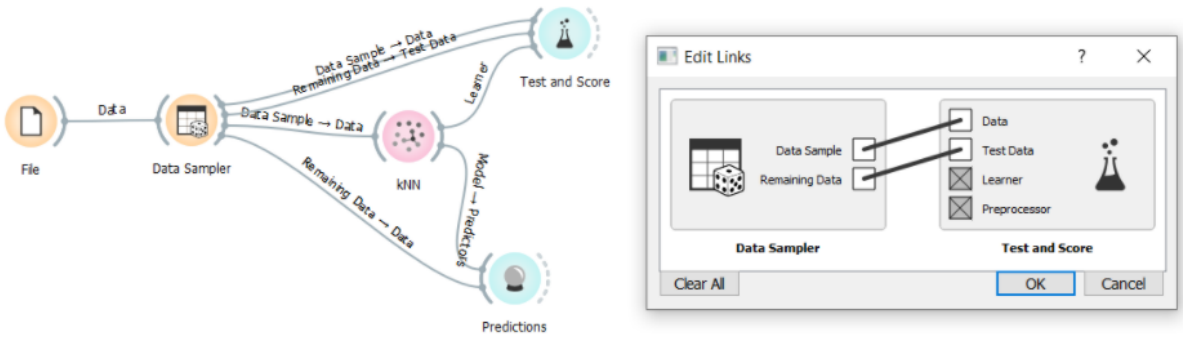
Double click the Predictions widget to see the predictions made by the model.



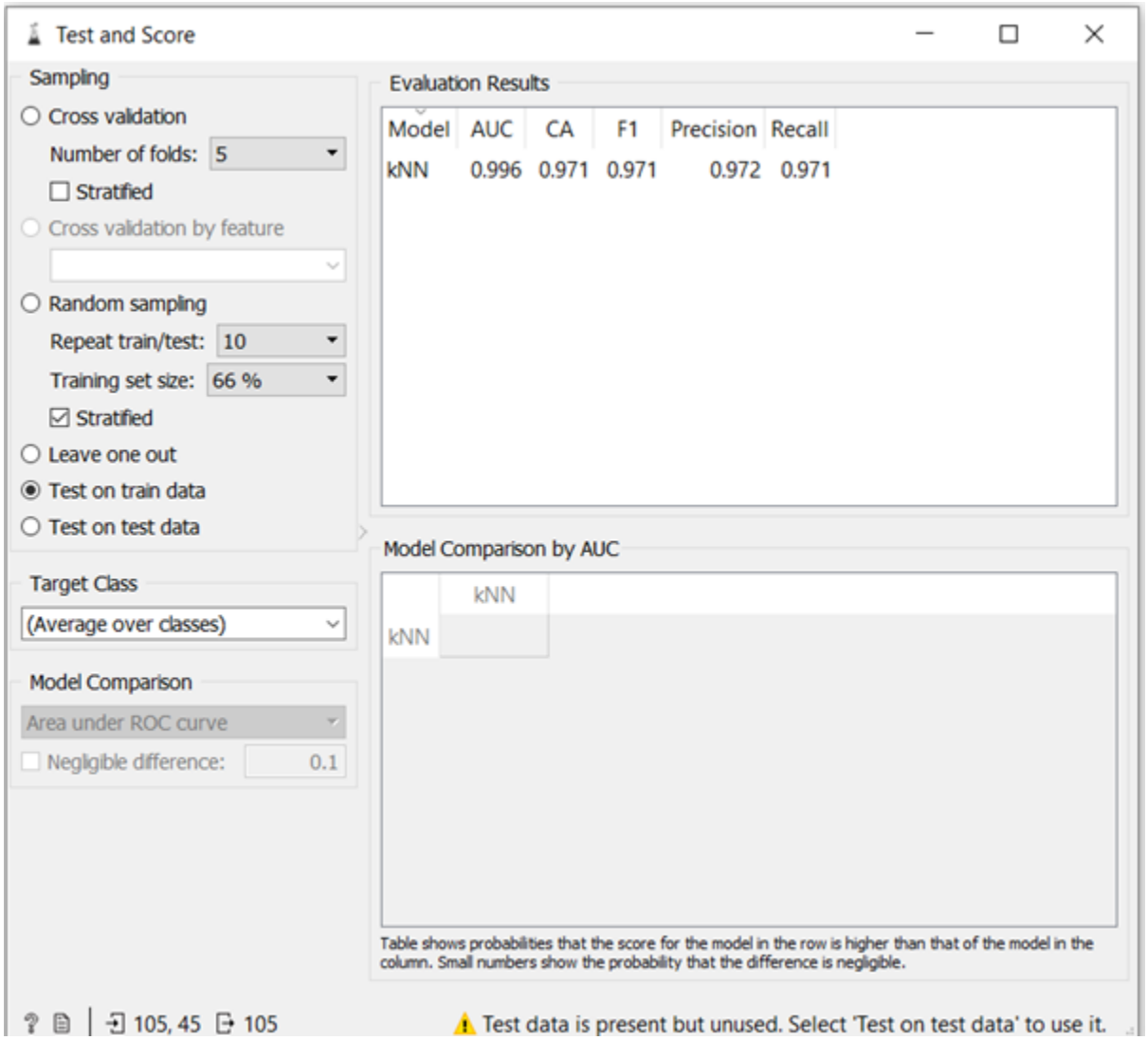
You can see there is a kNN column, followed by the columns from the test data. In the test data, we can see in the image above that the first flower was a setosa, with a sepal length of 4.6cm, and sepal width of 3.4cm, etc. The corresponding cell in the kNN column displays the probability of a flower with these measurements belonging to each class, according to the kNN model. For a more interesting example, look at instance 7: here we can see that the kNN model predicts a 0% probability that this instance is iris-setosa (the first class), a 20% chance that this instance is of the Iris-versicolor class (the second class), and an 80% chance that belongs to the Iris-virginica class (the third class), which it then predicts. As we can see, this prediction was correct.

As you can see, there are five different measurements of the model's performance at the bottom of the window. Let's focus on the CA (Classifier Accuracy for now). In this demonstration, we can see 97.8% of the predictions made on the test data were correct. Does your model also get 97.8%? If it doesn't, don't worry - remember that the 70-30 split of the data was random, and you will likely have trained and tested on slightly different parts of the dataset.

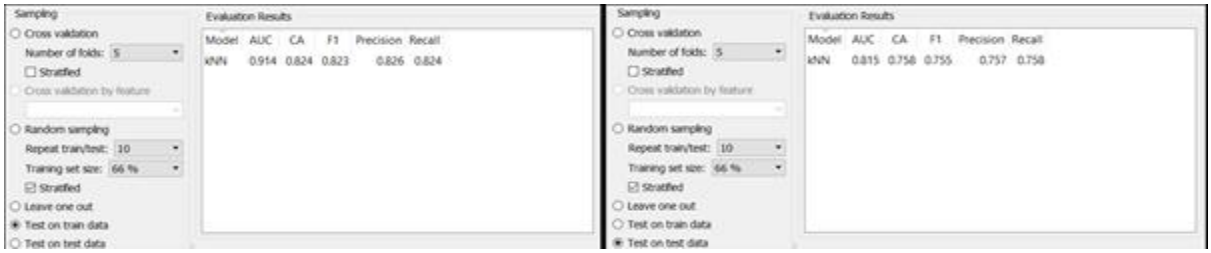
Let's have a look at how the model performs in more detail. Add a Test and Score widget, and ensure that the Data Sample is connected to the Data (telling Test and Score what data was used for training the model), and the Remaining Data is being used as Test Data.



Click on Test and Score, and you will see a number of different options for evaluating your kNN model. Begin by clicking on Test on train data, and inspecting the Classifier Accuracy (CA). Now try evaluating it on the test data (which your model has never seen before). They probably look very similar… perhaps the model is even better at predicting the instances which it has never seen before - this is strange! It's actually because the Iris data is very simple in comparison to most classification problems and we aren't seeing what normally happens in machine learning.

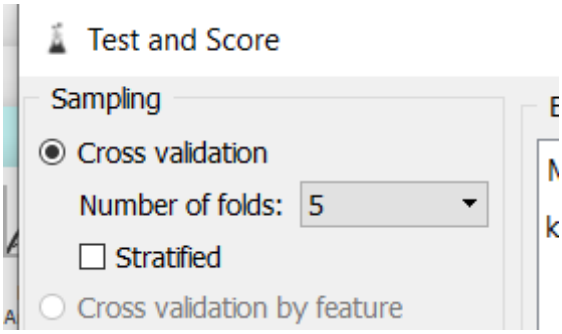


Let's try using a different dataset. From the File widget, change the file to the MotorClaims.csv that you downloaded before, and set the Claim variable's Role to the Target. Now, using the Test and Score widget, compare the kNN performance on the train and test data. With a more complicated problem, we can see that our model typically performs worse on unseen data - just as we would expect.



If we had just used our training data to measure how well we would expect our model to perform, we would have overestimated how good it is. Now, instead of 'test on train data', let's try Cross Validation - by using just the training data, we can use CV to evaluate our model more effectively.

Try 5 fold cross-validation, and compare the CV accuracy to the model's performance on unseen data. You will notice that CV gave us a much better estimation of the model's quality.



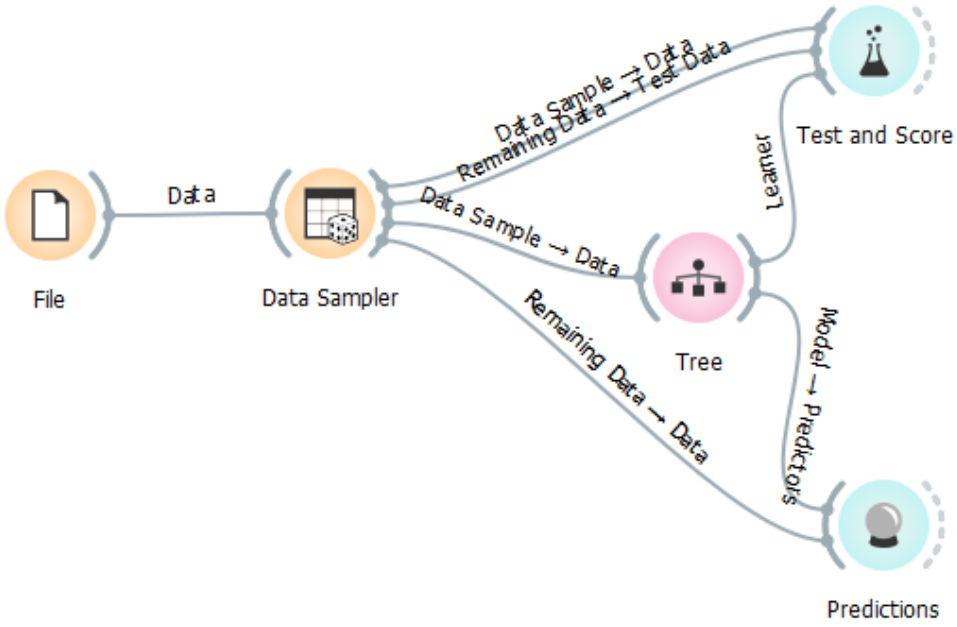
Now that we have a more reliable way to evaluate our model - let's try changing the parameters. Open the kNN widget and the Test and Score widget at the same time. Try changing the number of neighbours, the metric used to measure the distances between instances (default is Euclidean), and the weights of those distances. Each time you change one of these metrics, you will see the cross validation accuracy change. Experiment with different combinations until you get a higher score.

With these new parameters, test on the test data. Choosing the right parameters should have resulted in better performing models, even on data that your model has never seen before.

Decision Trees

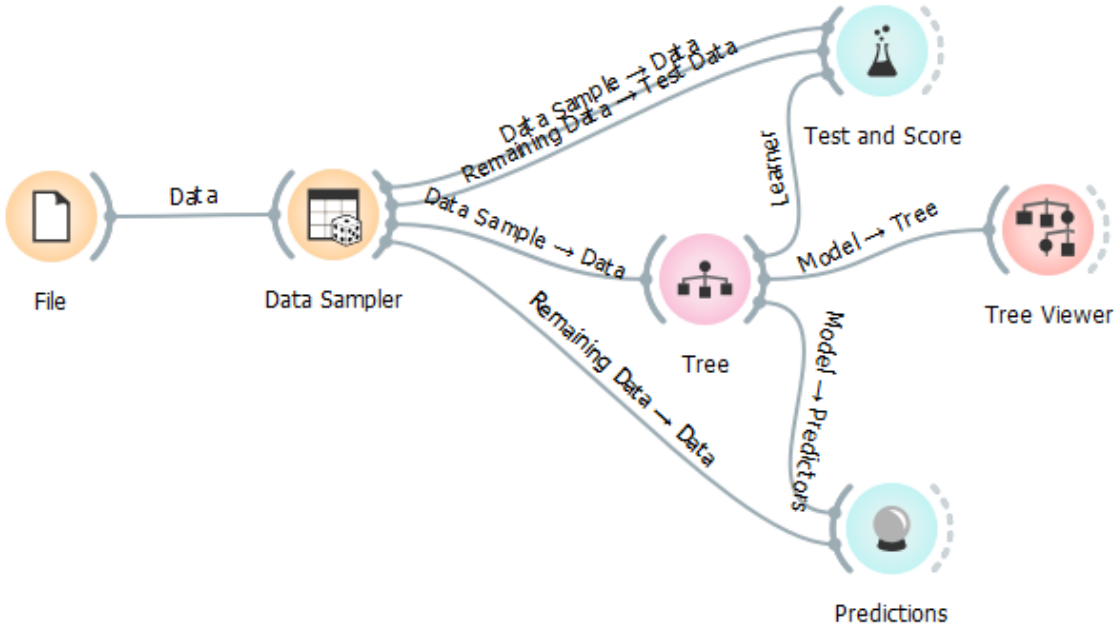
In this part, we are going to take the problem from earlier, and train and test a Decision Tree.

Just like in the previous exercise, create a File widget and load the iris dataset, setting iris as the target. Add a Data Sampler with a fixed proportion of 70% of the data, and add a Tree from the Models in the workspace. Connect it to the Data Sampler, Test and Score, and Prediction widgets just like we did with the kNN.

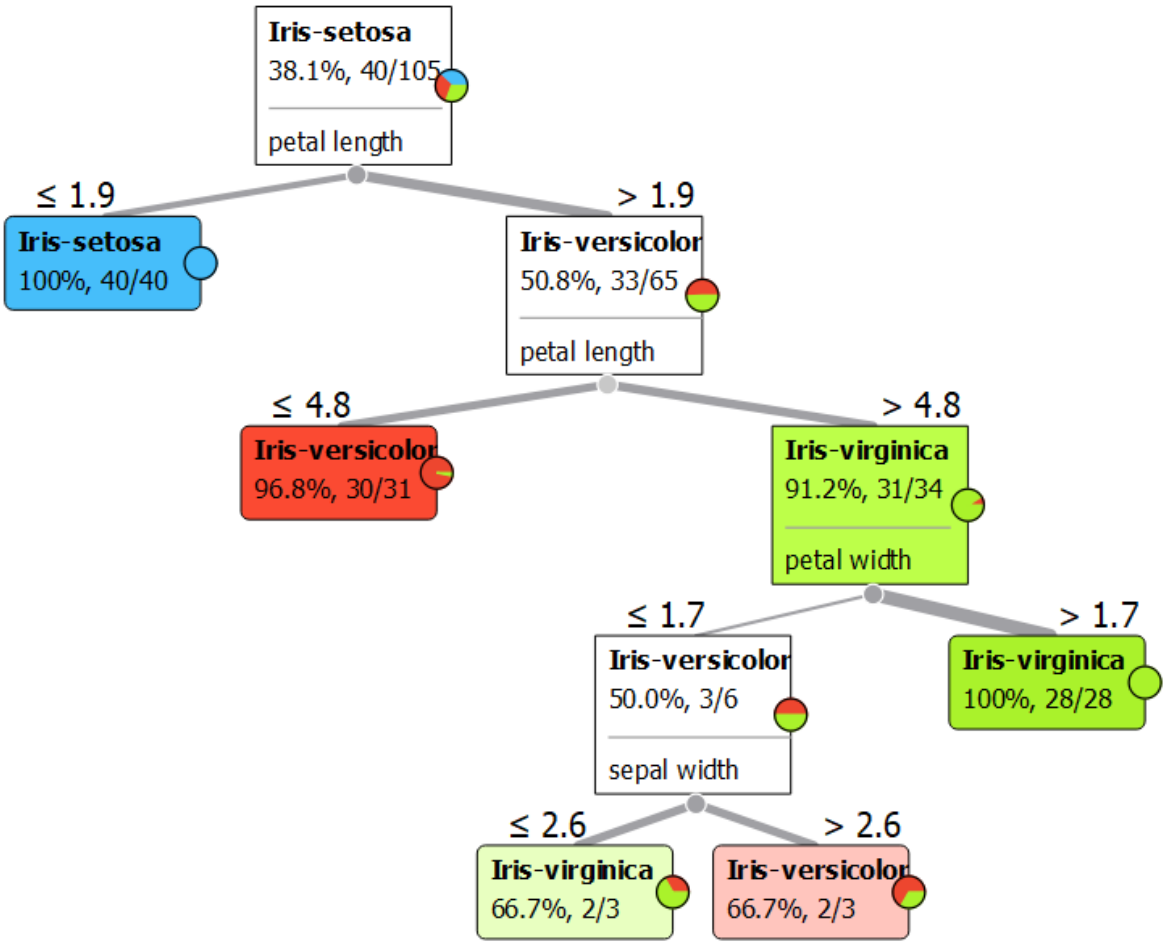


Explore the Test and Score widget. What better reflects the performance of the model on the unseen data (test data): testing on the training data, or cross validation on the training data?

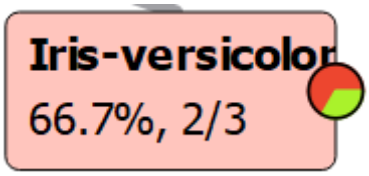
One of the most substantial advantages of Decision Trees are their explainability. Under Visualise, select Tree Viewer and connect it to the Decision Tree.



If you open the Tree Viewer, you will now be able to inspect the Decision Tree



Each internal node contains a variable name at the bottom, and a rule to split that variable on. In the example above, the first node (the root node) splits the data based on a petal length of 1.9cm. Each leaf node displays the predicted class of all instances that arrived at that node, and the percentage of those correctly identified.

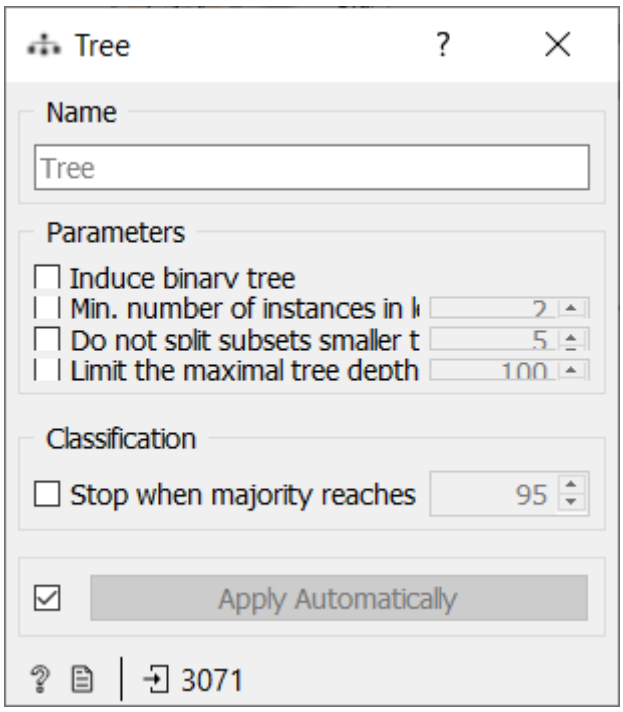


In the case of this node, 2 of the 3 flowers (66.7%) in this split of the dataset are Iris-versicolor.

As you can see, not all of the flowers are correctly identified, but the Tree stopped making further splits. This is to ensure that the model does not overfit.

Let's see what happens if we allow the tree to overfit - as before, change the dataset to MotorClaims.csv, and inspect the Tree Viewer. It's much more complicated now!

Check the Test and Score widget and make a note of the CA when testing on the train data, and when testing on the test data. Now click on the Tree and untick all of the boxes under Parameters and Classification.



These parameters are designed to stop the tree from creating overly complex rules that overfit to the training data and prevent generalisation to new datasets. Have a look at the Tree Viewer - you will see that every leaf node is now completely ‘pure’. Inspect the Test and Score widget and notice that the Test on train data is now perfect! (if it’s not, it’s simply a rounding error). However, when we test on the test data, we can see it has had a negative effect and the original model was better.