Predicting Rainfall in Australia: An End-to-End Machine Learning Project

Prepared by: Malek El-Benni

Date: 08/08/2025

Contents

1.0 Executive Summary

This report documents a comprehensive, end-to-end data science project focused on solving a critical business problem: predicting next-day rainfall in Australia. Leveraging a complex dataset of daily weather observations, this analysis moves from initial data exploration and rigorous preprocessing to the development and comparative evaluation of multiple machine learning models. The primary objective was to build a classifier that could accurately forecast the binary outcome of RainTomorrow.

The project successfully navigated significant data quality challenges, including numerous missing values and class imbalance. A multi-stage preprocessing pipeline was developed using KNIME to clean, transform, and engineer features, preparing the data for advanced modeling.

Five distinct classification algorithms were trained and evaluated: Decision Tree, Random Forest, K-Nearest Neighbours (KNN), Support Vector Machine (SVM), and a Multi-Layer Perceptron (MLP) Neural Network. All models were systematically assessed based on accuracy, precision, recall, and their performance on a private Kaggle competition. The Multi-Layer Perceptron (MLP) model was selected as the optimal solution, achieving the highest overall accuracy of 85.1% and the top Kaggle submission score of 0.75507. While all models demonstrated a bias towards the majority "No Rain" class due to dataset imbalance, the MLP provided the most balanced and effective performance.

2.0 Introduction & Project Objectives

In today's data-driven world, predictive analytics is an essential tool for forecasting and decision-making. This report addresses a critical data mining problem: accurately predicting the likelihood of rainfall on the next day, indicated by the RainTomorrow attribute. The ability to forecast rain is particularly valuable in fields such as agriculture, logistics, and urban planning, where weather conditions directly impact operational efficiency and resource allocation.

The project involves binary classification: determining whether it will rain (1) or not (0). Solving this necessitates not only a robust understanding of machine learning but also careful data preprocessing, feature engineering, and hyperparameter tuning to optimize model performance. This report documents the entire process, providing a principled and evidence-based approach to solving the problem.

3.0 Exploratory Data Analysis & Data Quality Assessment

The initial phase of the project focused on a deep exploration of the provided weather dataset to understand its structure, characteristics, and inherent challenges.

3.1 Data Profile

The dataset contains numerous meteorological attributes. A critical first step was to identify the data type of each attribute to inform subsequent analysis and preprocessing strategies. The attributes were categorized as follows:

Heading	Data Type	Reason
Date	Interval	Meaningful order and interpretable differences but lack a true zero.
Location	Nominal	Categorical data with no natural ordering.
MinTemp, MaxTemp	Ratio	Temperature in Celsius has a meaningful zero point (absolute zero).
Rainfall, Evaporation	Ratio	Possesses a true zero (0mm means no rainfall/evaporation).
Sunshine	Ratio	Measured in hours, with a true zero (no sunshine).
WindGustDir, WindDir9am/3pm	Nominal	Categorical wind directions with no inherent order.
WindGustSpeed, WindSpeed9am/3pm	Ratio	Possesses a true zero (no wind).
Humdity9am/3pm	Ratio	Percentage with a meaningful true zero (0% humidity).
Pressure9am/3pm	Ratio	Atmospheric pressure has a true zero (absolute vacuum).
Cloud9am/3pm	Ratio	Measured in eighths, with a true zero (0/8 means no clouds).
RainToday, RainTomorrow	Nominal	Binary categorical data (Yes/No) with no order.

3.2 Summary Statistics & Key Distributions

A statistical summary revealed significant data quality issues and interesting distributions that heavily influenced the project's direction.

☐	#	RowID	Column *String*	Min *Number (Float)*	Max *Number (Float)*	Mean *Number (Float)*	Std. deviation *Number (Float)*	Variance *Number (Float)*	Skewness *Number (Float)*	Kurtosis *Number (Float)*
☐	1	Rainfal	Rainfall	0	113.4	2.118	6.757	45.653	6.787	69.495
☐	2	Evapor	Evaporation	0	64.4	5.583	4.571	20.89	4.538	44.856
☐	3	Sunshi	Sunshine	0	13.8	7.602	3.802	14.457	-0.502	-0.794
☐	4	WindG	WindGustSpeed	9	117	40.275	13.594	184.791	0.955	1.812
☐	5	WindS	WindSpeed9am	0	56	13.997	8.931	79.757	0.745	0.759
☐	6	WindS	WindSpeed3pm	0	63	18.844	8.841	78.165	0.681	0.981
☐	7	Humidi	Humidity9am	1	100	68.691	19.038	362.457	-0.496	0.012
☐	8	Humidi	Humidity3pm	1	100	51.69	20.616	425.006	-0.022	-0.466
☐	9	Pressu	Pressure9am	989.5	1,040.1	1,017.639	7.138	50.953	-0.097	0.047
☐	10	Pressu	Pressure3pm	992.6	1,037.5	1,015.239	7.007	49.096	0.004	-0.184
☐	11	Cloud9	Cloud9am	0	8	4.54	2.88	8.292	-0.267	-1.508
☐	12	Cloud3	Cloud3pm	0	8	4.561	2.705	7.319	-0.231	-1.438

Overall sum *Number (Float)*	No. missings *Number (Integer)*	No. NaNs *Number (Integer)*	No. +∞s *Number (Integer)*	No. -∞s *Number (Integer)*	Median *Number (Float)*	Row count *Number (Integer)*	Histogram *Image (SVG)*
5,427.1	56	0	0	0	0	2618	
8,218.1	1146	0	0	0	4.8	2618	
10,155.9	1282	0	0	0	8.4	2618	
98,231	179	0	0	0	39	2618	
36,293	25	0	0	0	13	2618	
48,430	48	0	0	0	19	2618	
176,743	45	0	0	0	69	2618	
131,552	73	0	0	0	53	2618	
2,388,399.1	271	0	0	0	1,017.5	2618	
2,392,917.2	261	0	0	0	1,015.3	2618	
7,328	1004	0	0	0	5	2618	
7,074	1067	0	0	0	5	2618	

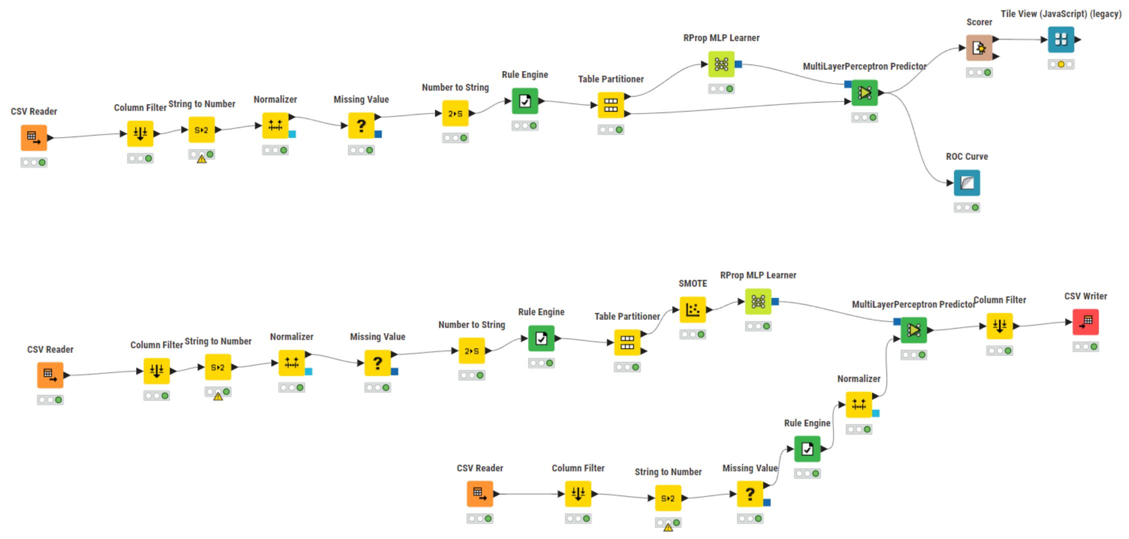
- **Significant Missing Data:** Several key predictive features suffered from a high volume of missing entries. Sunshine (1282 missing), Evaporation (1146 missing), Cloud9am (1004 missing), and Cloud3pm (1067 missing) were particularly problematic. This challenge necessitated a robust imputation strategy to avoid excessive data loss.
- **Highly Skewed Features:** Rainfall and Evaporation were found to be highly skewed to the right, with high kurtosis values. This indicates that most days are dry with low evaporation, but the dataset contains rare but extreme outlier events (heavy rainfall, high evaporation). This class imbalance is the central challenge of this predictive problem.
- **Temperature & Pressure:** Temperature attributes (MinTemp, MaxTemp) showed relatively normal distributions, while pressure attributes were very stable and tightly clustered around the mean, as expected.

3.3 Initial Findings from EDA

- **Wind Patterns:** Analysis of the nominal wind direction data indicated a predominant pattern of easterly gusts, with morning winds tending to be from the North and afternoon winds from the South.
- **Rainfall Frequency:** The modes for RainToday and RainTomorrow were both overwhelmingly "No," confirming the class imbalance suggested by the skewed Rainfall distribution.

4.0 Data Preprocessing for Machine Learning

Based on the EDA, a multi-stage preprocessing pipeline was developed in KNIME to prepare the data for modeling. The goal was to create a clean, complete, and correctly formatted dataset to maximize model performance.



4.1 Feature Selection

Several columns were excluded from the training dataset to reduce noise and multicollinearity:

- Location: While useful for EDA, the location itself was deemed less important for a generalized prediction model than the meteorological measurements (temperature, pressure, etc.).
- Evaporation, Sunshine: These columns were removed due to the excessive number of missing values. Imputing over 1000 missing entries could introduce significant bias and unreliable data.
- WindGustDir, WindDir9am, WindDir3pm: These categorical features were removed to simplify the initial models, which focused on numerical predictors.

4.2 Missing Value Imputation

An initial strategy of removing all rows with missing values proved unfeasible, as it led to unacceptable data loss. The final strategy involved imputation:

- **Numerical Columns:** Missing values were replaced with the **median** of their respective columns. The median was chosen over the mean to be more robust to the outliers identified during EDA.
- **Categorical Columns:** Missing values were replaced with the most frequent value (the **mode**).

4.3 Feature Transformation & Engineering

Several transformations were applied to make features suitable for machine learning algorithms:

- **Normalization (Z-Score):** All numerical features were standardized using Z-score normalization. This was a critical step to ensure that features with larger scales (like Pressure) did not unduly influence distance-based algorithms (KNN, SVM) or gradient-based optimization (MLP).
- **Discretization:** The continuous WindSpeed3pm attribute was discretized into four meaningful categories ("Slow Wind", "Medium Wind", "Fast Wind", "Very Fast Wind") based on its distribution, simplifying the feature for tree-based models.
- **Binarization:** The categorical RainToday feature ("Yes"/"No") was converted to a binary numerical format (1/0) using a Rule Engine node to serve as a valid input for the models.

4.4 Handling Class Imbalance (SMOTE)

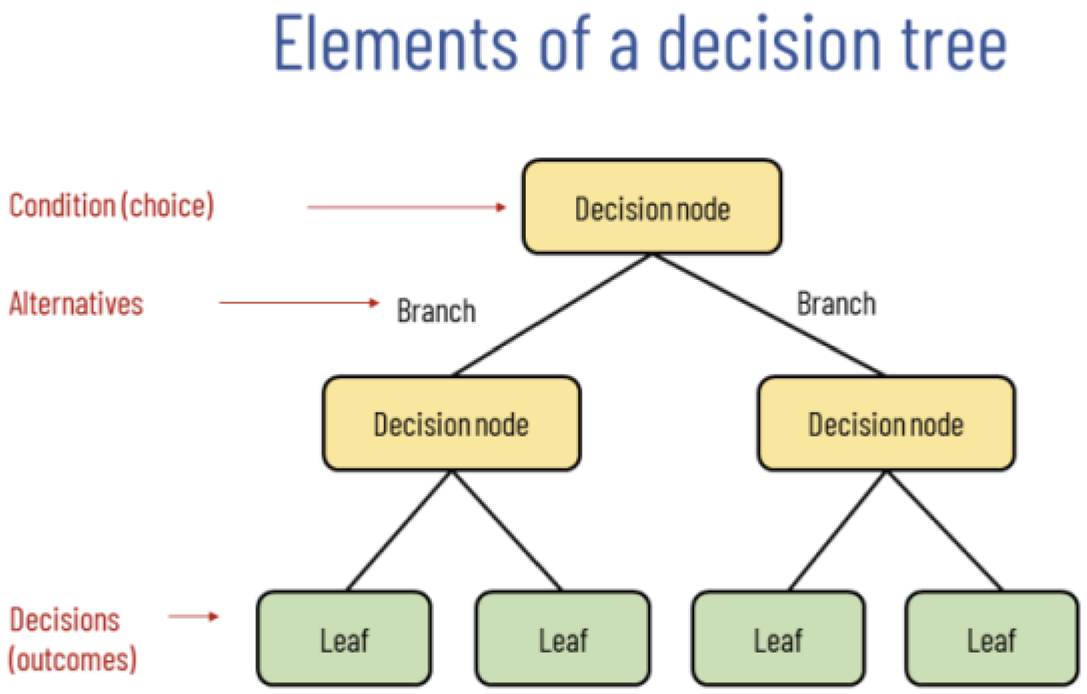
Given the significant imbalance between "Rain" and "No Rain" days, the **SMOTE (Synthetic Minority Over-sampling Technique)** was applied. This technique generates new synthetic samples for the minority class ("Rain") by interpolating between existing instances, creating a more balanced dataset for the models to train on. This was a crucial step to mitigate the models' natural bias towards predicting the majority "No Rain" class.

5.0 Predictive Modeling & Evaluation

Five different classification models were trained and evaluated to identify the best performer for this task. Each model was trained on a 70% partition of the data and tested on the remaining 30%.

5.1 Model 1: Decision Tree

A decision tree is a machine learning model used for classification and regression tasks. It resembles a flowchart-like structure where data is split at each node based on feature values, ultimately leading to a prediction at the tree's "leaves" or end nodes. Decision trees are popular due to their simplicity, interpretability, and ability to handle both numerical and categorical data.



Advantages of Decision Trees

- Easy to Interpret: Decision trees are visually intuitive and easy to understand, making them useful for explaining model decisions.
- Non-Linear Decision Boundaries: Decision trees can capture non-linear relationships between features and the target variable.
- No Need for Data Scaling: Decision trees do not require feature scaling (e.g., normalization or standardization), as they split based on thresholds.
- Handles Both Categorical and Numerical Data: Decision trees can handle both types of data without needing extensive preprocessing.

Disadvantages of Decision Trees

- Prone to Overfitting: Decision trees can easily overfit, especially when they grow deep and learn very specific patterns in the training data. This reduces their generalizability.
- High Variance: Small changes in the data can result in a very different tree structure, which can affect the model's stability.

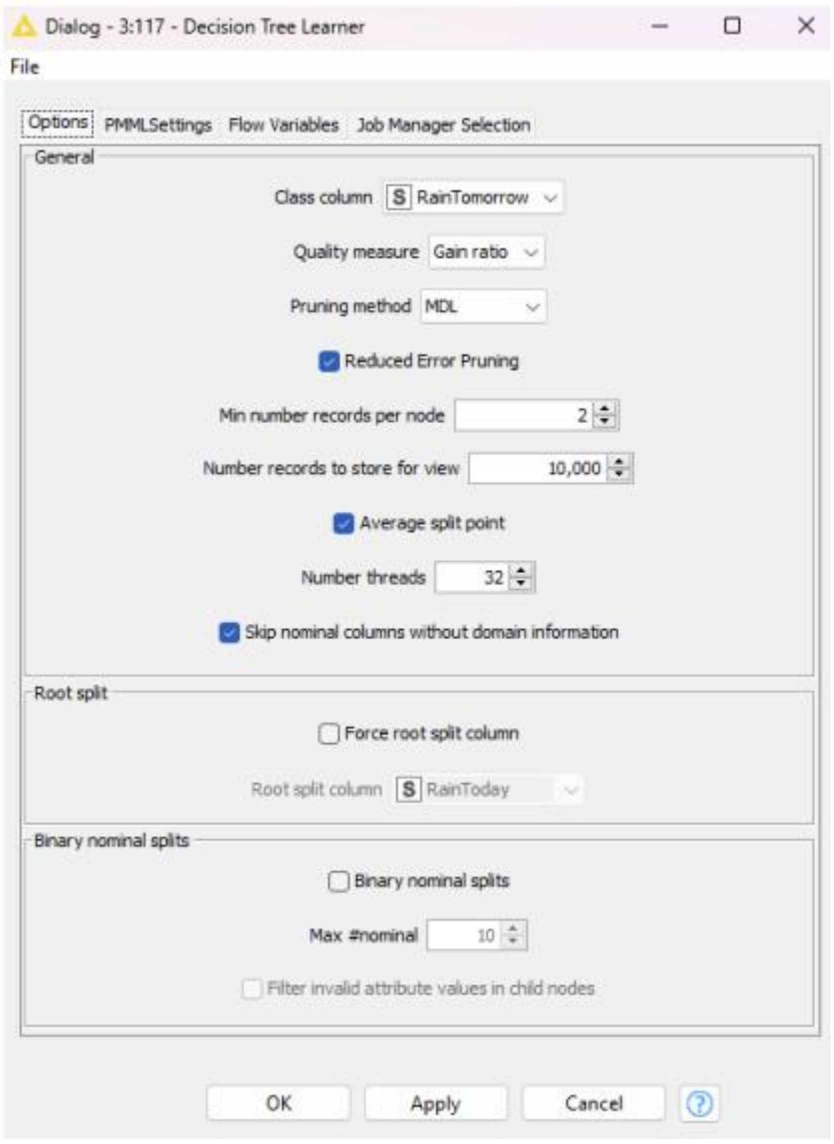
- Bias Toward Features with More Levels: In some cases, features with more unique values can dominate splits, leading to biased trees.

Key Features of the Decision Tree Learner Node:

1. Automatic Feature Selection:
    - The Decision Tree Learner node automatically selects the most relevant features at each split based on metrics that measure the "purity" or quality of the split. This selection process continues recursively down the tree.
    - Common split criteria include Gini Impurity, Entropy/Information Gain, and Variance Reduction (for regression tasks).
2. Configurable Stopping Criteria:
    - The node allows you to control the complexity of the tree by setting maximum depth, minimum number of records per node, or minimum information gain to prevent overfitting.
    - By adjusting these parameters, you can control how "deep" or complex the tree becomes, balancing the trade-off between model accuracy and interpretability.
3. Handles Both Numerical and Categorical Data:
    - The Decision Tree Learner node can work with both numerical and categorical data, making it versatile and adaptable to various data types. No extensive preprocessing is needed, as decision trees handle different data types natively.
4. Binary and Multi-Class Classification:
    - This node supports binary classification (e.g., "Yes" vs. "No") as well as multiclass classification, allowing you to predict outcomes with multiple classes.
5. Visualization of the Tree Structure:
    - After training, you can use the Decision Tree View node or the Decision Tree Predictor node to visualize the resulting tree structure, making it easy to interpret and explain the model’s decision-making process.

The Decision Tree Learner node is configured to predict the "RainTomorrow" class column, using Gain ratio as the quality measure for selecting splits, and MDL (Minimum Description Length) as the pruning method to simplify the tree and prevent overfitting. Reduced Error Pruning is enabled, which helps further reduce complexity by trimming branches that don’t improve accuracy. A minimum of 2 records per node is set, allowing the tree to grow until each node has at least two data points. The configuration limits 10,000 records for view, uses average split points to handle splits on continuous features, and assigns 32 threads to parallelize the computation, which can speed up model training. Additionally, nominal columns without domain information are skipped, meaning categorical columns without predefined levels are ignored. The "Root split" and "Binary nominal splits" options are left unchecked, meaning the tree will automatically determine the root split and handle nominal splits according to default settings. This configuration is optimized for performance and interpretability, balancing model complexity with generalization.

Configuration of the Decision Tree Learner Node:


Dialog - 3:117 - Decision Tree Learner

File

Options | PMMLSettings | Flow Variables | Job Manager Selection

General

Class column: S RainTomorrow

Quality measure: Gain ratio

Pruning method: MDL

[✓] Reduced Error Pruning

Min number records per node: 2

Number records to store for view: 10,000

[✓] Average split point

Number threads: 32

[✓] Skip nominal columns without domain information

Root split

[ ] Force root split column

Root split column: S RainToday

Binary nominal splits

[ ] Binary nominal splits

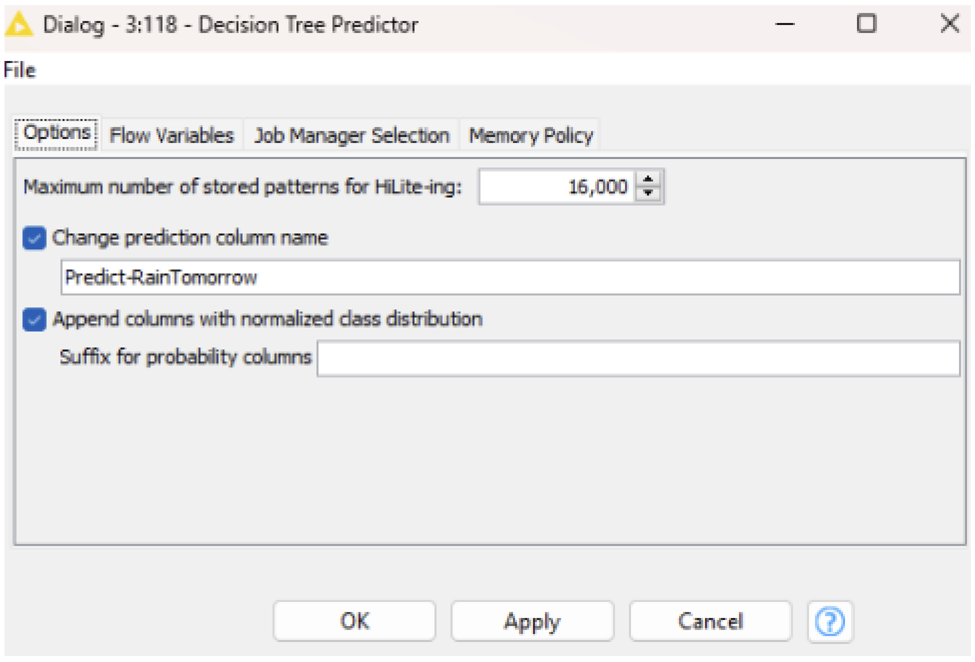
Max #nominal: 10

[ ] Filter invalid attribute values in child nodes

To use the trained decision tree model, we must use a “Decision Tree Predictor” node. The Decision Tree Predictor node is used to apply a previously trained decision tree model to new or unseen data to generate predictions. This node takes the decision tree model created by the Decision Tree Learner node and applies it to a test dataset, assigning predicted class labels (for classification tasks) or continuous values (for regression tasks) based on the rules learned during training. The key features of this node are:

1) Model Application:

    a) The node uses the trained decision tree to predict outcomes on new data by following the learned decision rules from the root to a leaf node, based on the feature values of each data point.
2) Classification or Regression Predictions:
    a) For classification, it assigns the most common class label in the leaf node. For regression, it outputs the average value of the target variable in the leaf node.
3) Probability Output (Optional):
    a) If the decision tree model was trained for classification, the Decision Tree Predictor node can also provide the probability of each class for each prediction, giving a sense of confidence in the model's output.
4) Compatibility with Other Models:
    a) This node works exclusively with decision tree models built in KNIME (typically from the Decision Tree Learner node), ensuring consistent integration within KNIME workflows.
5) Efficient Handling of Large Datasets:
    a) The Decision Tree Predictor node can handle large datasets efficiently, leveraging the learned tree structure to make quick predictions without reevaluating the model.


Dialog - 3:118 - Decision Tree Predictor

Maximum number of stored patterns for HiLite-ing: 16,000

☑ Change prediction column name

Predict-RainTomorrow

☑ Append columns with normalized class distribution

Suffix for probability columns:

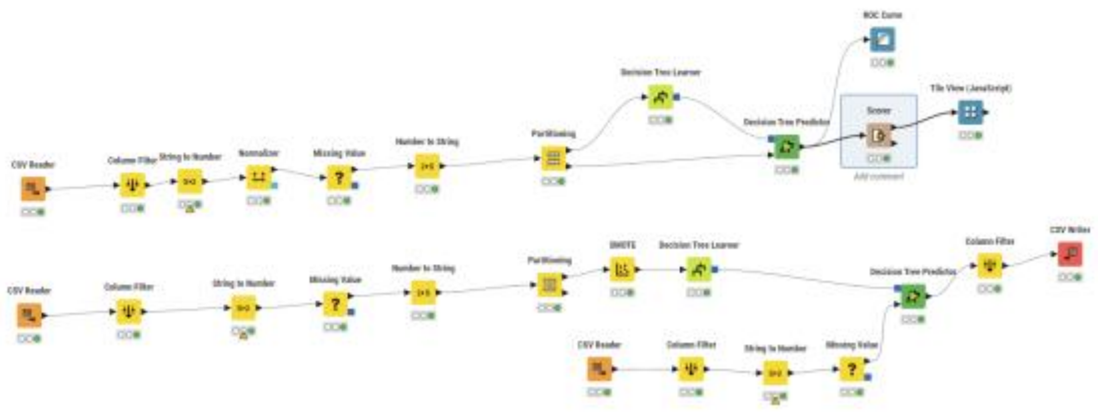
This Decision Tree Predictor node configuration is set up to apply the trained decision tree model and customize the output prediction columns. Listed below is a rundown of each setting:

1) Maximum number of stored patterns for HiLite-ing:
    a) This field is set to 16,000, which specifies the maximum number of data points (patterns) the node will store for interactive HiLite-ing, a feature in KNIME that highlights data points across views. This setting is generally useful for datasets

with large numbers of rows, ensuring efficient memory usage. This has been set to 16,000 to cover all the datapoints present in our dataset.

2) Change prediction column name:
   a) This option is enabled and set to "Predict-RainTomorrow". It renames the prediction output column to "Predict-RainTomorrow," making it clear that this column contains the predicted values for the "RainTomorrow" target variable. Customizing the name helps differentiate it from the original target variable and improves readability in the final output. Additionally, the name has been changed to align with the required format for the Kaggle submissions.
3) Append columns with normalized class distribution:
   a) This option is checked, which means the node will append additional columns representing the normalized probability distribution for each class. For example, if the target variable has classes like "Yes" and "No," this setting will output the probability of each class for each data point, showing the model's confidence in its predictions. This will be useful for later use in the "ROC curve" node.
4) Suffix for probability columns:
   a) This field is left blank, meaning that no suffix will be added to the probability columns. If you enter a suffix here (e.g., "_Prob"), the probability columns will be labelled with this suffix appended to the class names (e.g., "Yes_Prob" and "No_Prob").

For all the models, I split training/evaluation and actual use of the model into separate areas. For example:



The top workflow is for training/evaluating the accuracy and capabilities of the model, and the bottom workflow is for finding the "RainTomorrow" value of the "UnknownData" csv.

Regarding accuracy, we use the "scorer" node and the "ROC Curve" node to evaluate the model we have created. Regarding our model's accuracy, this was measured using the "scorer" node.

Scorer

The Scorer node in KNIME is a tool used to evaluate the performance of a classification model by comparing the predicted class labels with the actual class labels in the test data. This node generates key evaluation metrics that help assess the effectiveness of a model in accurately predicting outcomes, especially in classification tasks. The key features of the "scorer" node are:

1. Confusion Matrix:
    - The Scorer node produces a confusion matrix, which summarizes the model's performance by showing the counts of correct and incorrect predictions for each class. It includes:
        - True Positives (TP): Correctly predicted positive instances.
        - True Negatives (TN): Correctly predicted negative instances.
        - False Positives (FP): Incorrectly predicted positive instances (Type I error).
        - False Negatives (FN): Incorrectly predicted negative instances (Type II error).
    - The confusion matrix provides a detailed breakdown of how well the model distinguishes between different classes.
2. Classification Metrics:
    - Based on the confusion matrix, the Scorer node calculates various classification metrics, including:
        - Accuracy: The overall percentage of correct predictions.
            - Formula: (TP+TN)/(TP+TN+FP+FN)
        - Precision: The ratio of true positives to all predicted positives, showing how often the model is correct when it predicts a positive outcome.
            - Formula: TP/(TP+FP)
        - Recall (Sensitivity): The ratio of true positives to all actual positives, indicating the model's ability to identify positive cases.
            - Formula: TP/(TP+FN)
        - F1 Score: The harmonic means of precision and recall, providing a balanced measure of model performance, especially useful for imbalanced datasets.

- Cohen's Kappa is an additional metric provided by the Scorer node that measures the level of agreement between the predicted and actual classifications, adjusted for agreement occurring by chance. It ranges from -1 to 1:
    - ❖ 1 indicates perfect agreement between predictions and actual labels.
    - ❖ 0 indicates no agreement beyond what is expected by chance.
    - ❖ Negative values indicate less agreement than expected by chance.
- • Cohen's Kappa is especially useful for imbalanced datasets, as it provides a more reliable measure of model performance by accounting for random chance.

3. Support for Multi-Class Classification:
    - o The Scorer node can handle multi-class classification problems, computing metrics for each class individually as well as overall metrics, such as weighted averages across all classes.

Using the scorer node, the following matrix was generated:

#	RowID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives
1	0	11376	2020	1414	550
2	1	1414	559	11376	2020
3	Overall	12790	2579	12790	2570
	Percentage (%)	41.630	8.394	41.630	8.365

Based on this table, we can see that the accuracy of the model is approximately, 0.833. This is supported by using the formula previously mentioned, (TP+TN)/(TP/TN+FP+FN), where:

TP = 12790

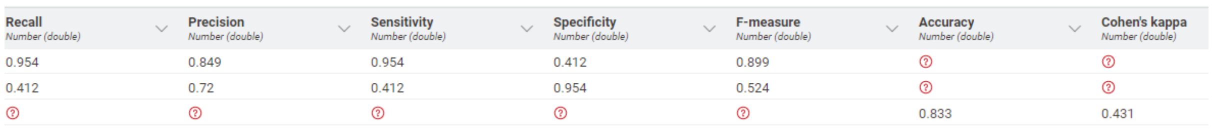
FP = 2579

TN = 12790

FN = 2570

(12790+12790)/(12790+12790+2579+2570) = 0.83

As shown above, we generally must calculate the values for the metrics ourselves, however, the scorer node places these metrics in a table for us as shown below:



Recall Number (double)	Precision Number (double)	Sensitivity Number (double)	Specificity Number (double)	F-measure Number (double)	Accuracy Number (double)	Cohen's kappa Number (double)
0.954	0.849	0.954	0.412	0.899	?	?
0.412	0.72	0.412	0.954	0.524	?	?
?	?	?	?	?	0.833	0.431

In the table above, we can see the values for recall, precision, sensitivity, specificity, F-measure, total accuracy and Cohen’s kappa shown for each column.

An accuracy of 0.833 is generally considered a “good” or “adequate” result, however, this truly depends on the context of the problem, the dataset, and the specific use case. Some factors to consider when interpreting accuracy include:

Type of Problem and Baseline Accuracy

- If the problem has a high baseline accuracy (e.g., classifying something with an imbalance where 80% of cases belong to one class), 83% might not be much better than simply predicting the majority class. In these cases, other metrics like precision, recall, F1 score, or Cohen's Kappa are more informative.
- For balanced datasets, an accuracy of 83% suggests the model is correctly classifying most cases, which could be acceptable or even impressive.

2. Class Imbalance

- If the dataset is imbalanced (e.g., one class is much more frequent than the other), accuracy can be misleading because the model might be "accurate" primarily by predicting the majority class correctly. In these cases, you’d want to check metrics like precision, recall, and F1 score for each class or look at a confusion matrix to see where the model may be underperforming on minority classes.

3. Business Impact and Acceptable Error Rate

- In some fields, an 83% accuracy may be sufficient, while in others, it may be inadequate. For instance:
    - In medical diagnostics or fraud detection, even a few misclassifications can be critical, so higher accuracy (or sensitivity) may be required.
    - For recommendation systems or sentiment analysis, 83% might be more than sufficient, as occasional errors may not have severe consequences.
- Assessing the cost of misclassification in the specific use case can help determine if 83% is acceptable.

4. Alternative Metrics for a Fuller Evaluation

- Accuracy alone may not tell the full story. Evaluating other metrics can provide more insights:
    - Precision and Recall: For binary classification, these can help understand the model's performance on each class.
    - F1 Score: Balances precision and recall, offering a single metric that may be more informative in imbalanced datasets.
    - Cohen's Kappa: Gives a chance-adjusted measure of agreement, which can be helpful if the classes are imbalanced or if there is a high baseline accuracy.

5. Comparison to Other Models or Benchmarks

- Compare the 83% accuracy to:
    - Benchmark Models: If other models (or a simple baseline) perform similarly or better, the model may not be effective enough.
    - Human-Level Performance: If humans achieve higher accuracy on the same task, the model may need improvement.
    - Previous or Alternative Models: If 83% represents an improvement over previous models or methods, it may be a good outcome.

However, given the dataset and tools available, I believe 83% to be a sufficient result, given the fact that there were:

- Many missing datapoints
- Time constraints
- Low impact of misclassification

ROC Curve

The ROC Curve node is a tool for evaluating the performance of binary classification models. ROC stands for Receiver Operating Characteristic, and the ROC curve is a graphical representation of a model's ability to discriminate between the positive and negative classes across different threshold settings. It provides insight into the model's true positive rate (sensitivity) and false positive rate at various classification thresholds, helping you understand the trade-off between correctly identifying positive cases and incorrectly identifying negative cases as positive.

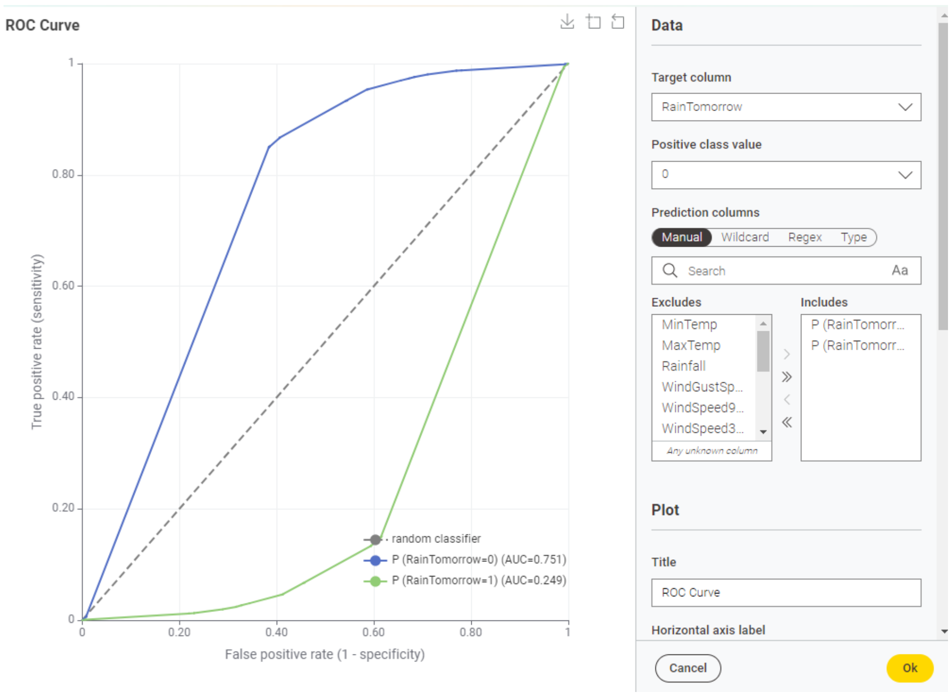
Key Features of the ROC Curve Node

1. Generates ROC Curve:
    - The node plots the ROC curve by varying the decision threshold and calculating the true positive rate (TPR) and false positive rate (FPR) at

each threshold. The curve illustrates the model's performance across different thresholds.

2. Calculates AUC (Area Under the Curve):
    - The node also calculates the AUC (Area Under the Curve), a single-value metric that represents the overall performance of the model. An AUC of 1.0 indicates perfect classification, while an AUC of 0.5 indicates no discriminatory power (equivalent to random guessing).
    - The AUC score provides a summary of the model's ability to correctly rank positive instances higher than negative ones, with higher AUC values indicating better performance.
3. Threshold Analysis:
    - The ROC Curve node allows users to explore different thresholds for classification, making it possible to identify the optimal threshold based on specific needs (e.g., maximizing sensitivity or minimizing false positives).
4. Visual Comparison of Models:
    - If you have multiple models, the ROC Curve node allows you to overlay multiple ROC curves in a single plot, enabling visual comparison of each model's performance.

ROC curve 1:


ROC Curve
True positive rate (sensitivity)
False positive rate (1 - specificity)
0
0.20
0.40
0.60
0.80
1
random classifier
P (RainTomorrow=0) (AUC=0.751)
P (RainTomorrow=1) (AUC=0.249)
Cancel
Ok

Data

Target column: RainTomorrow

Positive class value: 0

Prediction columns: Manual | Wildcard | Regex | Type

Search

Excludes: MinTemp, MaxTemp, Rainfall, WindGustSp..., WindSpeed9..., WindSpeed3..., Any unknown column

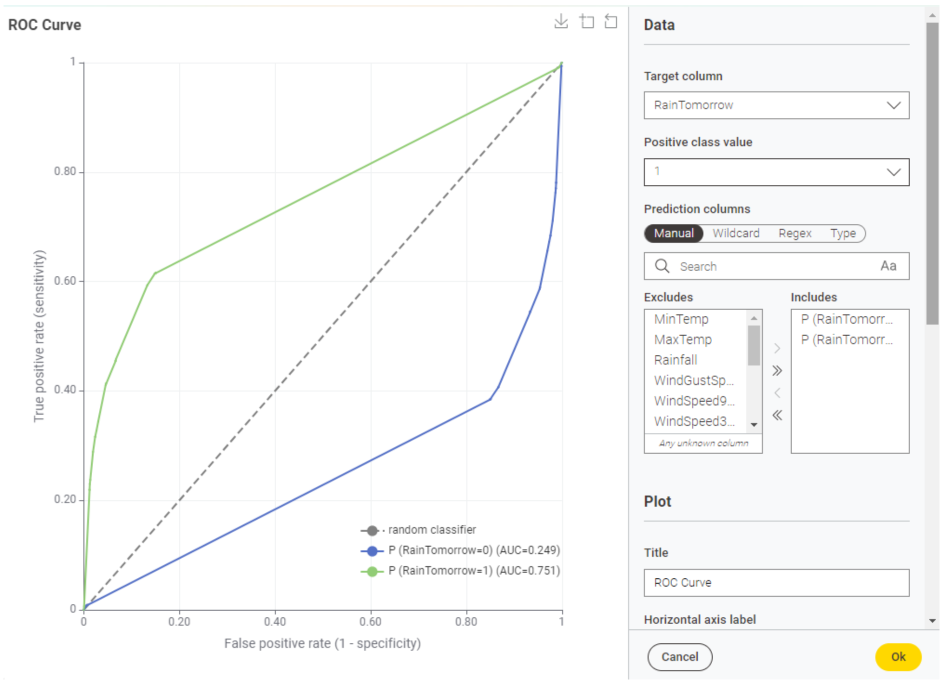
Includes: P (RainTomorr..., P (RainTomorr...

Plot

Title: ROC Curve

Horizontal axis label

ROC Curve 2:


ROC Curve
True positive rate (sensitivity)
False positive rate (1 - specificity)
random classifier
P (RainTomorrow=0) (AUC=0.249)
P (RainTomorrow=1) (AUC=0.751)
0
0.20
0.40
0.60
0.80
1
Cancel
Ok

Data

Target column: RainTomorrow

Positive class value: 1

Prediction columns: Manual | Wildcard | Regex | Type

Excludes: MinTemp, MaxTemp, Rainfall, WindGustSp..., WindSpeed9..., WindSpeed3...

Includes: P (RainTomorr..., P (RainTomorr...

Plot

Title: ROC Curve

Horizontal axis label

The decision tree classifier managed to achieve an accuracy of 0.833 on the testing data. This suggests that the performance of the model is good. The ROC curves suggest a class imbalance with non-rainy days being 0.751 and rainy days being 0.249. The issue arises with the minority classes being ungeneralised. This imbalance allowed for weaker results on rainy days and stronger results for the non-rainy days (majority). In essence, this means the model is biased for non-rainy days.

Analysis of the metrics revealed high precision and recall for non-rainy days, however, showed lower precision and recall for rainy days. This proves the bias of the classifier.

5.2 Model 2: Random Forest

Random Forest Learning is an ensemble machine learning method that combines multiple decision trees to create a more accurate and robust model. It's particularly useful for both classification and regression tasks, especially when working with large, complex, or noisy datasets. Developed by Leo Breiman and Adele Cutler, random forests build a "forest" of decision trees that work together to improve predictive performance and reduce overfitting, which individual decision trees are prone to.

Random forest models are effective because they:

- Reduces Overfitting: Individual decision trees tend to overfit to the training data, especially if they are deep. Random forests reduce this overfitting by averaging multiple trees, which creates a more generalized model.
- Handles High Dimensionality and Noisy Data: Random forests work well with datasets that have many features or noisy data, as each tree only uses a subset of features, and noise gets averaged out over the many trees.

- Robust to Missing Data: Random forests can handle missing values and maintain accuracy, as each tree only needs a subset of data and features.
- Feature Importance: Random forests can provide a measure of feature importance, helping identify which features are most influential in predicting the target variable.

Random Forest learning works by doing the following:

1. Ensemble of Decision Trees:
    a. A random forest consists of many decision trees (often hundreds or thousands), each trained on a slightly different version of the data. Each tree in the forest makes its own prediction, and the final prediction of the random forest is determined by aggregating the predictions of all trees.
    b. For classification tasks, the forest "votes" on the final class by taking the majority vote from all trees. For regression tasks, the final prediction is usually the average of all tree outputs.

2. Bootstrapping:
    a. Random forest uses a technique called bootstrapping to create diversity among the trees. During training, each tree is built on a different subset of the original data, sampled with replacement (meaning some instances may be used more than once, while others may not be used at all). This creates diverse trees that learn slightly different patterns, reducing the risk of overfitting to specific data points.

3. Random Feature Selection (Feature Bagging):
    a. At each split within a tree, only a random subset of features is considered, rather than all features. This is known as feature bagging, and it further reduces correlations between the trees, as each tree considers different features at different splits.
    b. This random selection of features prevents any single feature from dominating the model and promotes diversity, which improves the ensemble's robustness and generalization ability.

4. Aggregating Predictions:
    a. Once all trees are built, the random forest combines their predictions. In classification, it takes the majority vote, so the most frequently predicted class across all trees is chosen as the final prediction. In regression, the predictions from all trees are averaged to produce the final output.
    b. By combining the predictions of multiple trees, random forests achieve higher accuracy and are less sensitive to noise or overfitting compared to single decision trees.

Random Forest Learner

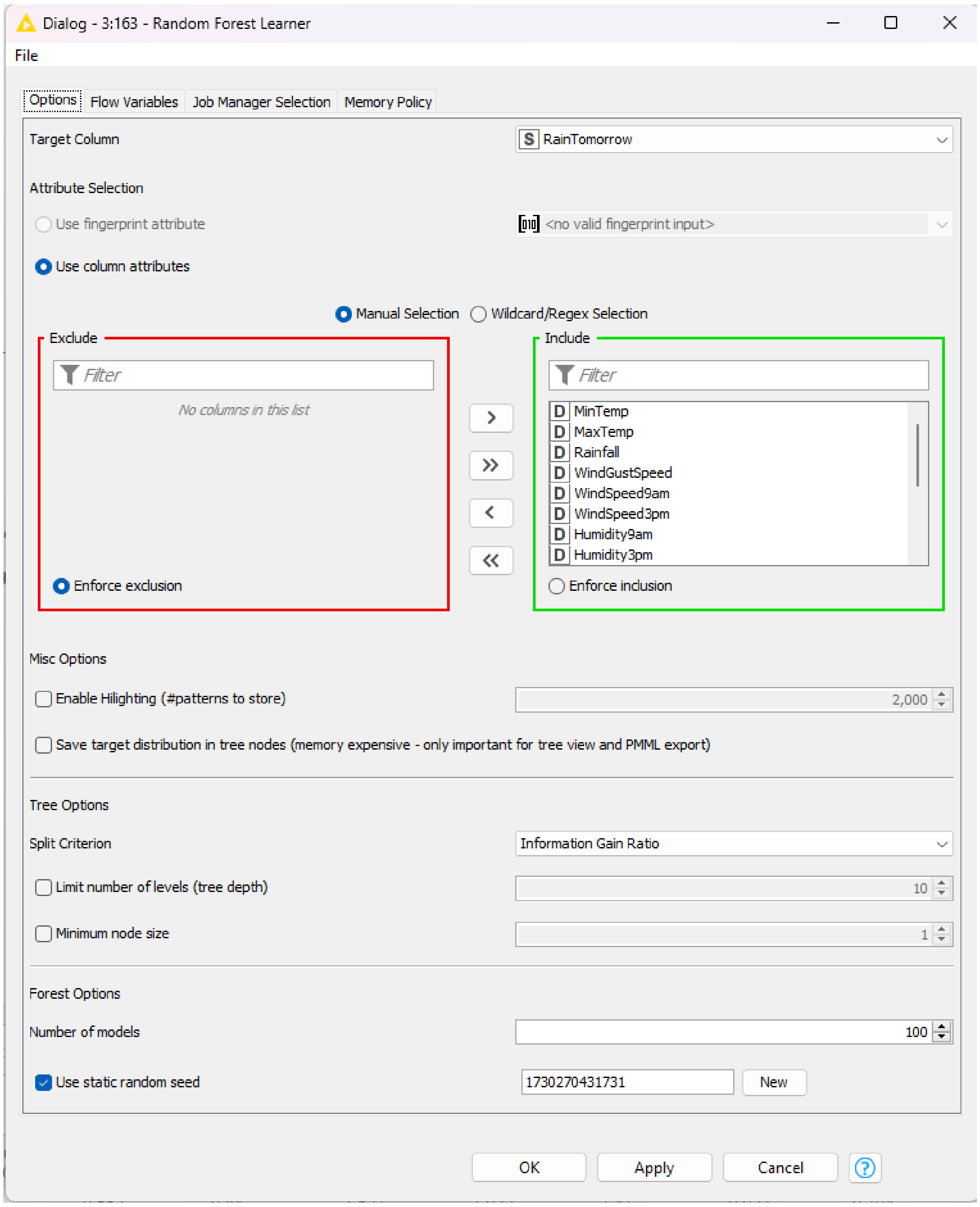
The Random Forest Learner node in KNIME is a tool used to train a random forest model. Random forests build multiple decision trees and combine their predictions to create a more robust and accurate model. This node is often used for both classification and regression tasks and is particularly powerful for datasets with complex patterns, high dimensionality, or noisy data. The Key Features of the Random Forest Learner Node are:

1. Ensemble of Decision Trees:
    - Random forests are made up of multiple decision trees (often hundreds), each trained on a different subset of the data. Each tree provides a "vote" for the final prediction, and the model aggregates these votes to make a robust prediction.
    - For classification, the final prediction is typically the most common class across the trees (majority voting), while for regression, the final prediction is the average of all trees' outputs.
2. Bootstrapping and Random Sampling of Features:
    - The Random Forest Learner node uses bootstrapping, where each tree is trained on a random sample (with replacement) of the training data.
    - Additionally, at each split within a tree, only a random subset of features is considered, which reduces correlations between trees and improves the overall generalization of the model. This feature sampling is particularly helpful when there are many irrelevant or redundant features.
3. Variable Importance:
    - The Random Forest Learner node can provide information on feature importance, showing which features are the most influential in the model's predictions. This helps in understanding the key drivers in the data and can aid in feature selection and interpretability.
4. Handles High Dimensionality and Noisy Data:

- Random forests perform well on high-dimensional datasets (datasets with many features) and are robust to noise, as they build multiple trees and combine their results, reducing the impact of any individual outlier or noisy feature.

5. Configurable Parameters:
    - The node provides various settings to control the model, including:
        - Highlight
        - Split Criterion
        - Node Size
        - Number of trees: Controls how many decision trees to build. More trees generally improve accuracy but also increase computation time.
        - Maximum tree depth: Limits the depth of each tree, which can help control overfitting.
        - Minimum number of records per leaf: Controls the minimum number of data points needed in a leaf node, which can also help reduce overfitting.

Configuration of Random Forest Learner Node:


Dialog - 3:163 - Random Forest Learner

File

Options | Flow Variables | Job Manager Selection | Memory Policy

Target Column: S RainTomorrow

Attribute Selection

○ Use fingerprint attribute <no valid fingerprint input>

● Use column attributes

● Manual Selection ○ Wildcard/Regex Selection

Exclude

Filter

No columns in this list

● Enforce exclusion

Include

Filter

D MinTemp
D MaxTemp
D Rainfall
D WindGustSpeed
D WindSpeed9am
D WindSpeed3pm
D Humidity9am
D Humidity3pm

○ Enforce inclusion

Misc Options

☐ Enable Hilighting (#patterns to store) 2,000

☐ Save target distribution in tree nodes (memory expensive - only important for tree view and PMML export)

Tree Options

Split Criterion: Information Gain Ratio

☐ Limit number of levels (tree depth) 10

☐ Minimum node size 1

Forest Options

Number of models: 100

☑ Use static random seed 1730270431731 New

OK | Apply | Cancel

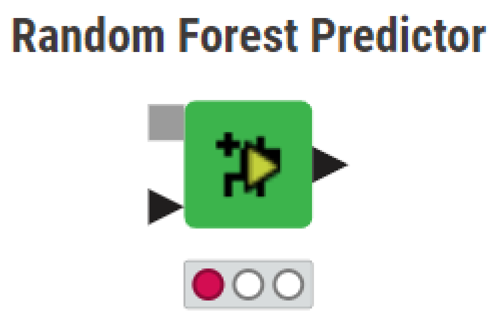
This node is configured to build a predictive model for the "RainTomorrow" target variable using a selection of weather-related features. The target column, set as "RainTomorrow," is the variable the model will aim to predict, based on the included input features. The Attribute Selection is set to use column attributes, with specific features manually selected in the Include section on the right, which lists features such as "MinTemp," "MaxTemp," "Rainfall," "WindGustSpeed," and various wind speed and humidity measurements taken at different times of the day. This configuration ensures that the model only uses these weather-related predictors, while the Exclude section on the left is empty, meaning no variables are explicitly excluded from the analysis. The

"Enforce exclusion" and "Enforce inclusion" options are selected, solidifying the chosen features to include or exclude, preventing any automatic adjustments that might inadvertently affect the attribute set.

In terms of Tree Options, the model is set to use the Information Gain Ratio as the split criterion. This criterion measures the improvement in information (or reduction in impurity) each split provides, helping the model identify the most informative splits for each decision tree. No constraints are set on the tree depth or the minimum node size, which gives the model flexibility to grow trees fully without restrictions on complexity, potentially enhancing predictive performance but also increasing the risk of overfitting, which the ensemble approach of random forests generally mitigates. Under Forest Options, the "Number of models" parameter is set to 100, meaning the random forest will consist of 100 decision trees, each contributing to the final prediction through majority voting (for classification) or averaging (for regression). A larger forest generally improves accuracy and robustness, as more trees help balance out individual tree biases, although it also increases computational demands.

A static random seed is enabled, set to a specific value, which ensures the reproducibility of the model results. By using a static seed, the random sampling processes within the random forest (such as bootstrapping the data and selecting random subsets of features at each split) will produce the same results each time this configuration is executed, making the model behaviour consistent across multiple runs. Miscellaneous options include settings for highlighting and saving target distributions in tree nodes, though neither is enabled. The highlighting option allows for storing patterns for visualization, while saving target distributions in tree nodes is memory-intensive and is typically used for model interpretability in the tree view, especially when exporting to PMML (Predictive Model Markup Language). Overall, this configuration balances flexibility in tree growth with control over input features, ensuring reproducibility, and aims to build a robust model by aggregating the predictions of 100 diverse trees based on an informative subset of weather features.

Random Forest Predictor:



The Random Forest Predictor node is used to apply a trained random forest model to new data to generate predictions. It takes the model built by the Random Forest Learner node and uses it to predict outcomes for unseen data, either by majority voting (for classification) or averaging (for regression). This node outputs the predicted class labels or values, along with optional probabilities for each class in classification tasks, allowing you to assess the model's predictions on new datasets.

Configuration of random forest predictor:

This Random Forest predictor configuration is set up to generate clear and informative predictions for the target variable "RainTomorrow". This is a quick rundown of the configuration:

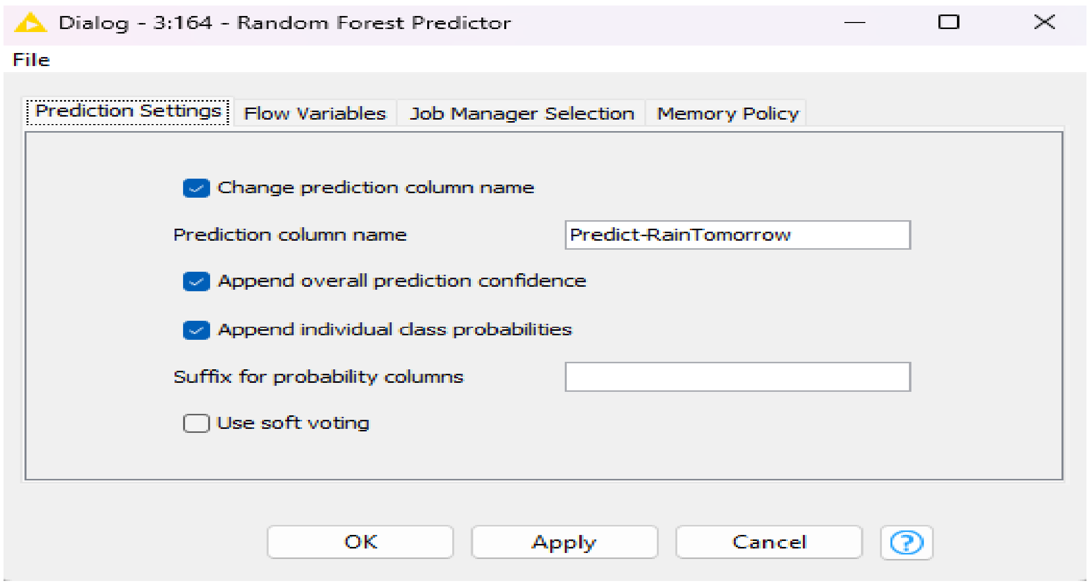
- Change Prediction Column Name:
  - This option is enabled, with the prediction column name set to "Predict-RainTomorrow." This custom name clearly distinguishes the prediction output from the actual target variable, making it easier to identify in the results table and avoid confusion during analysis. This was also done to accommodate the Kaggle submission requirements.

- Append Overall Prediction Confidence:
  - This setting is checked, meaning that the output will include an overall confidence score for each prediction. This confidence score provides insight into how certain the model is about each prediction, which is valuable for assessing reliability, especially when decisions are based on the model's output.

- Append Individual Class Probabilities:
  - By enabling this option, the predictor appends probabilities for each class. This is especially helpful for understanding the likelihood of each class, allowing for more nuanced decision-making. For instance, users might set a custom threshold based on probability rather than using strict binary predictions, providing flexibility depending on the application's risk tolerance.

- Suffix for Probability Columns:
  - This field is left blank, meaning no additional suffix will be added to the probability column names. While this is optional, leaving it blank keeps the output tidy and straightforward, using default naming conventions for probabilities.

- Use Soft Voting:
    - This option is unchecked, so the predictor uses hard voting, where each tree votes on a single class, and the final prediction is the majority vote. Hard voting is typically effective in random forests for classification tasks, providing a decisive prediction without the need for averaging probabilities. Soft voting (if enabled) would average the probabilities across trees, which is useful in cases with highly imbalanced classes or when confidence scores are essential, but it is generally not needed in well-balanced datasets.


Dialog - 3:164 - Random Forest Predictor

File

Prediction Settings | Flow Variables | Job Manager Selection | Memory Policy

✓ Change prediction column name

Prediction column name: Predict-RainTomorrow

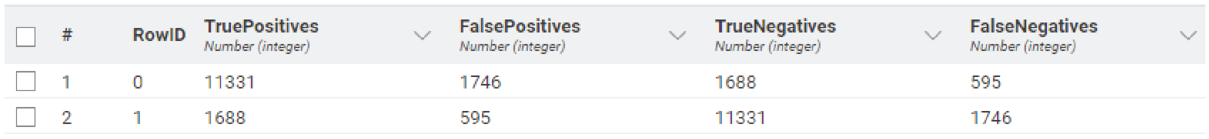
✓ Append overall prediction confidence

✓ Append individual class probabilities

Suffix for probability columns:

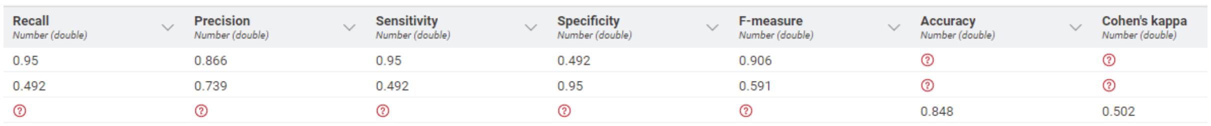
☐ Use soft voting

Using the scorer node, we generated the following confusion matrix:



#	RowID	TruePositives Number (integer)	FalsePositives Number (integer)	TrueNegatives Number (integer)	FalseNegatives Number (integer)
1	0	11331	1746	1688	595
2	1	1688	595	11331	1746

Using these results, we can calculate the following metrics:



Recall Number (double)	Precision Number (double)	Sensitivity Number (double)	Specificity Number (double)	F-measure Number (double)	Accuracy Number (double)	Cohen's kappa Number (double)
0.95	0.866	0.95	0.492	0.906	?	?
0.492	0.739	0.492	0.95	0.591	?	?
?	?	?	?	?	0.848	0.502

1. Recall:
    a. The recall values are 0.95 and 0.492 for the two classes, indicating that the model is very good at correctly identifying one class (likely the positive class, if we're predicting rain) with a recall of 0.95, but only manages to identify 49.2% of instances for the other class. This imbalance suggests that the model is more sensitive to one class, potentially leading to biased predictions if both classes are equally important.

2. Precision:

   a. The precision values are 0.866 and 0.739, showing that when the model predicts each respective class, it is correct 86.6% and 73.9% of the time. These are reasonable precision values, with the higher precision for the class with better recall. This indicates that, for one class, the model not only captures most of the relevant cases but also avoids too many false positives. However, for the other class, while precision is decent, it's less effective in recall, which affects the overall balance.

3. Sensitivity (Recall) and Specificity:
   a. Sensitivity is aligned with recall values, with 0.95 for one class and 0.492 for the other. Specificity, which measures the true negative rate, is 0.492 for one class and 0.95 for the other. This reinforces the observation that the model is better at detecting one class than the other. Ideally, for a balanced performance, both sensitivity and specificity should be closer in value.

4. F-measure:
   a. The F-measure values are 0.906 and 0.591. The high F-measure for one class (0.906) reflects a good balance between precision and recall, showing that the model performs well on this class. The lower F-measure (0.591) for the other class indicates a weaker balance, mainly due to the lower recall, meaning the model struggles to fully capture this class while maintaining precision.

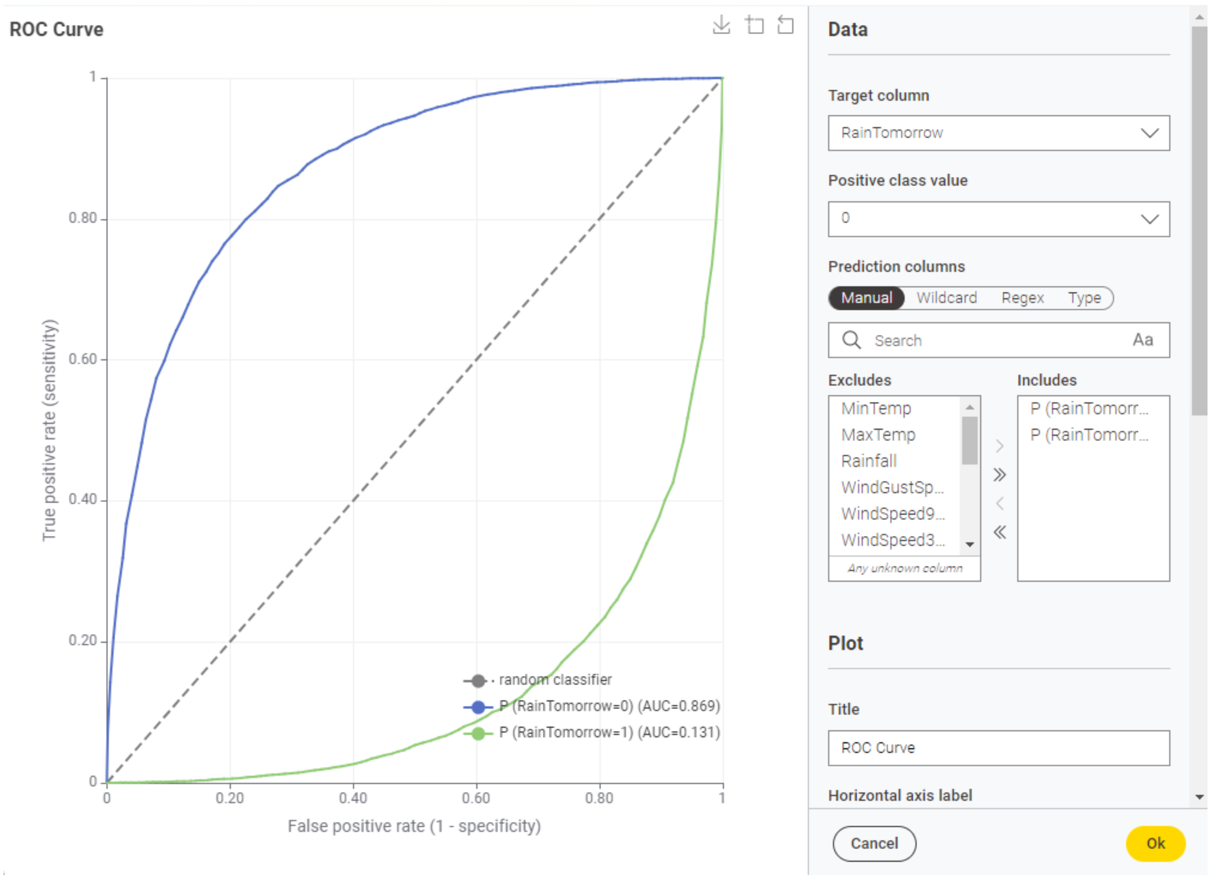
5. Accuracy:
   a. The overall accuracy of the model is 0.848 (84.8%), which indicates that the model is correct in its predictions about 85% of the time. While this seems acceptable, accuracy alone can be misleading in imbalanced datasets or when one class performs better than the other. Given the variation in recall, the model may appear more accurate due to its performance on the dominant class.

6. Cohen's Kappa:
   a. Cohen's Kappa is 0.502, which provides a chance-adjusted measure of agreement between the model predictions and actual values. A Kappa of 0.502 is moderate, indicating that while the model performs better than random chance, there is room for improvement in achieving a better balance across classes. Ideally, a Kappa closer to 1 would indicate a strong, reliable model, whereas a value around 0.5 suggests moderate performance.

Overall, this model demonstrates strong performance for one class, with high recall, precision, and F-measure values, but it struggles with the other class, as seen in its lower recall, F-measure, and sensitivity. The high accuracy (84.8%) is encouraging but may be somewhat misleading due to class imbalance, as one class is identified much better than the other. Cohen's Kappa of 0.502 suggests moderate agreement, meaning the model is better than random but could be more balanced. To improve, consider techniques like adjusting class weights, tuning hyperparameters, or exploring additional features to increase sensitivity for the weaker class and achieve a more balanced performance across all metrics.

ROC Curve

ROC Curve 1:


ROC Curve
1
0.80
0.60
0.40
0.20
0
True positive rate (sensitivity)
0
0.20
0.40
0.60
0.80
1
False positive rate (1 - specificity)
random classifier
P (RainTomorrow=0) (AUC=0.869)
P (RainTomorrow=1) (AUC=0.131)
Manual
Wildcard
Regex
Type
Search
Aa
Cancel
Ok

Data

Target column: RainTomorrow

Positive class value: 0

Prediction columns

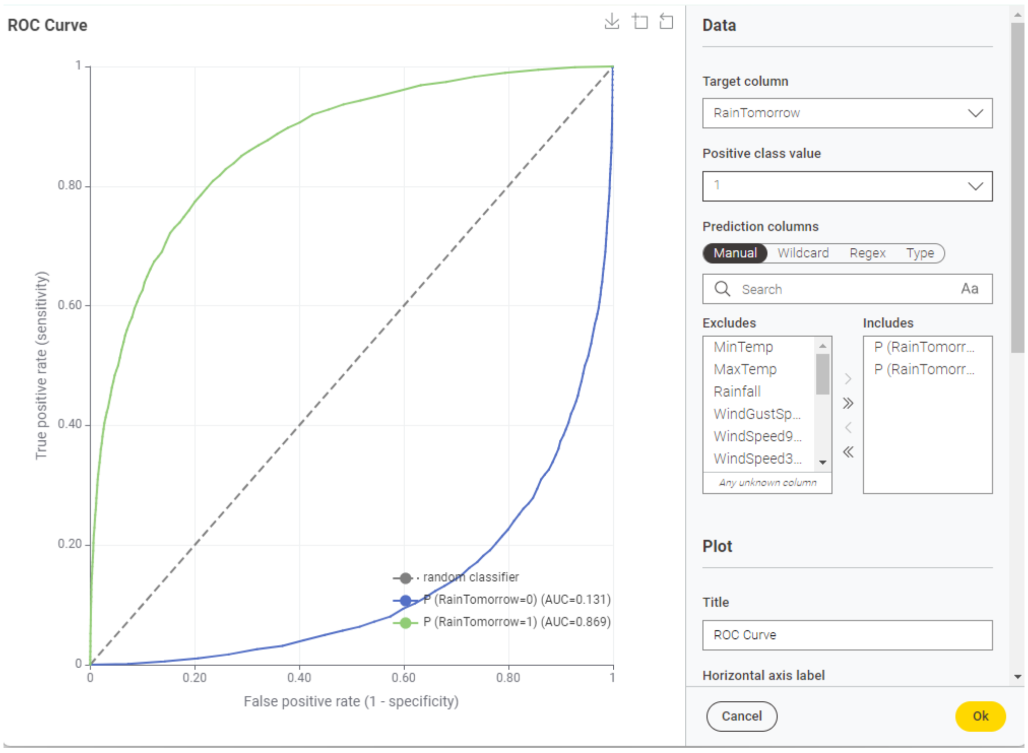
Excludes	Includes
MinTemp	P (RainTomorr...
MaxTemp	P (RainTomorr...
Rainfall	
WindGustSp...	
WindSpeed9...	
WindSpeed3...	
Any unknown column	

Plot

Title: ROC Curve

Horizontal axis label

ROC Curve 2:


ROC Curve
1
0.80
0.60
0.40
0.20
0
True positive rate (sensitivity)
0
0.20
0.40
0.60
0.80
1
False positive rate (1 - specificity)
random classifier
P (RainTomorrow=0) (AUC=0.131)
P (RainTomorrow=1) (AUC=0.869)
Manual
Wildcard
Regex
Type
Search
Aa
Cancel
Ok

Data

Target column: RainTomorrow

Positive class value: 1

Prediction columns

Excludes	Includes
MinTemp	P (RainTomorr...
MaxTemp	P (RainTomorr...
Rainfall	
WindGustSp...	
WindSpeed9...	
WindSpeed3...	
Any unknown column	

Plot

Title: ROC Curve

Horizontal axis label

This ROC curve evaluates the performance of the Random Forest model for predicting that it will rain tomorrow (the "1" class in "RainTomorrow"). With an AUC of 0.131, this model exhibits extremely poor performance for identifying rain instances. An AUC of 0.5 would indicate no discriminative power (equivalent to random guessing), and an AUC of 0.131 suggests the model is far worse than random for this class. In other words, the model fails to reliably distinguish between rainy and non-rainy days when predicting the positive class ("1" for rain), which is a significant limitation if rain prediction is crucial.

The blue curve demonstrates the model's performance with non-rainy days (majority). The majority classes are clearer more prominent in our dataset; thus, the model has an easier time predicting the major values instead of the minor values.

Additionally, this model's confusion matrix shows that for minority classes, the precision and recall are lower, whilst, for the majority classes (non-rainy days), the results were much higher. This shows that our model is biased against minorities. This model achieved a Kaggle score of 0.75429.

5.3 Model 3: K-Nearest Neighbors (KNN)

K-Nearest Neighbours (K-NN) is a simple, yet powerful, supervised machine learning algorithm used for classification and regression tasks. K-NN is an example of an instance-based or "lazy" learning algorithm, meaning it does not create a model in the traditional sense but instead makes predictions based on the stored training examples. KNN works by:

1. Storing Training Data:
- In K-NN, the entire training dataset is stored, and no actual "model" is created during the training phase. This is why it's known as a "lazy" learning algorithm—it only performs computation when making predictions.

2. Predicting New Instances:
- When a new data point (query instance) needs to be classified or have a value predicted, the algorithm:
- Measures the distance between the query point and all points in the training dataset. The most common distance metric is Euclidean distance for continuous data, although other metrics like Manhattan or Minkowski distances can also be used.
- Identifies the "k" closest points (neighbours) to the query instance. Here, "k" is a user-defined parameter that specifies how many neighbours to consider for making the prediction.
- Classifies or predicts the query point based on the values of these neighbours.

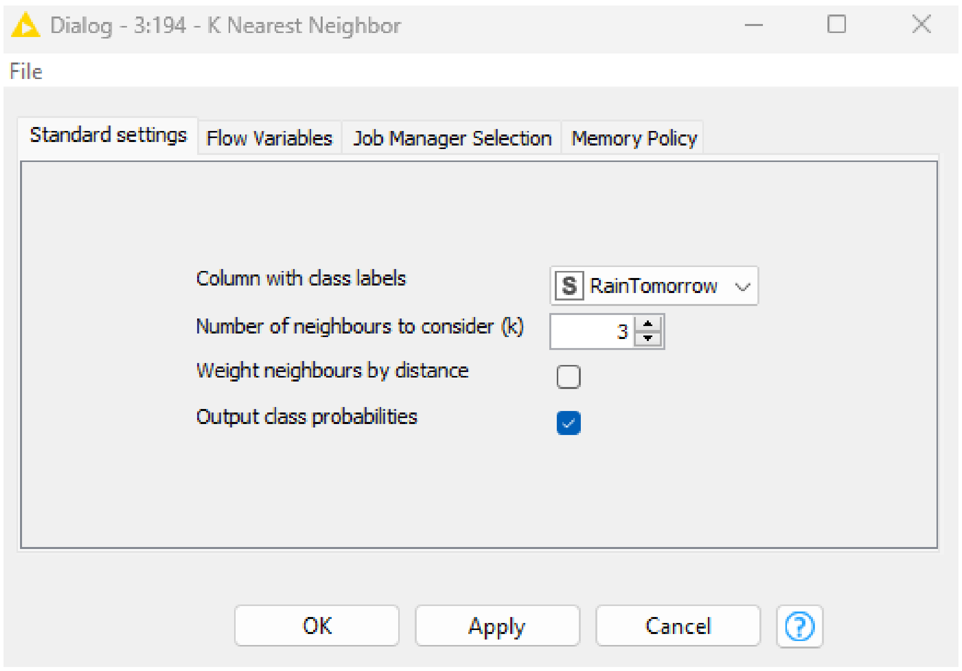
3. Decision Rules for Prediction:

- Classification: For classification, K-NN assigns the query instance to the class that is most common among its "k" nearest neighbours (majority voting). For example, if "k" is 5 and three of the nearest neighbours are labelled "Class A" while two are labelled "Class B," the algorithm would classify the new instance as "Class A."
- Regression: For regression, K-NN takes the average (or sometimes the median) of the values of the k nearest neighbours to predict the value for the query instance.

K-Nearest Neighbour (KNN) Learner

The KNN Learner node in KNIME is a tool that configures the K-Nearest Neighbours (K-NN) algorithm for classification or regression tasks. The KNN Learner node essentially saves the configuration of the algorithm, stores the dataset, and prepares it for use in predictions when combined with the KNN Predictor node.

Configuration of KNN learner node:



Column with class labels: RainTomorrow
Number of neighbours to consider (k): 3
Weight neighbours by distance: ☐
Output class probabilities: ☑

This configuration is setup to use 3 nearest neighbours to make predictions. The list below is the reasoning for each selected option:

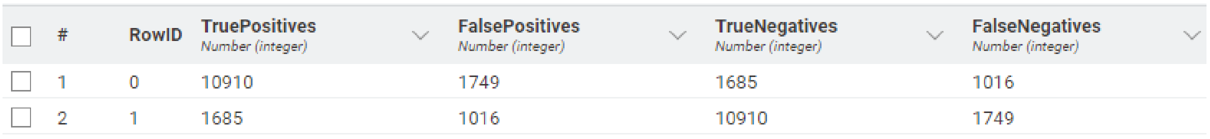
- Column with Class Labels:
  - The target column, labelled as "RainTomorrow," is selected as the class label. This means that the KNN model will attempt to predict whether it will rain tomorrow (the target variable) based on the values in this column.

- Number of Neighbours to Consider (k):
  - The number of neighbours (k) is set to 3, meaning that when predicting a new instance, the algorithm will look at the three closest data points in the feature space to determine the outcome. Setting k=3 allows the

model to make decisions based on a small group of similar instances, which can be beneficial for capturing local patterns but may be more sensitive to noise than larger values of k.

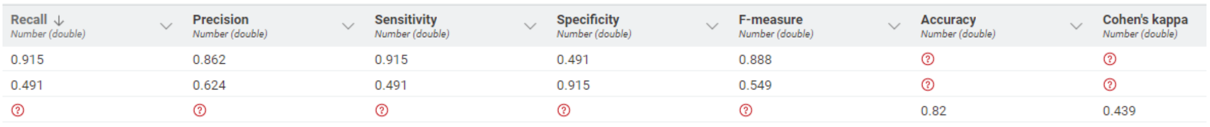
- Weight Neighbours by Distance:
    - This option is unchecked, meaning that each of the 3 neighbours will contribute equally to the prediction, regardless of how close or far they are from the new instance. If this option were enabled, closer neighbours would have more influence on the prediction than those farther away, which can sometimes improve accuracy by prioritizing closer matches. In this case, the model assumes that all three neighbours are equally relevant.
- Output Class Probabilities:
    - This option is checked, so the model will output the probabilities for each class (e.g., probability of rain or no rain) in addition to the predicted class label. This is useful for understanding the model's confidence in each prediction and allows for more nuanced decision-making, such as setting custom thresholds or using probabilities for further analysis.

Combining this configuration with a scorer node, generates the following confusion matrix:



#	RowID	TruePositives Number (integer)	FalsePositives Number (integer)	TrueNegatives Number (integer)	FalseNegatives Number (integer)
1	0	10910	1749	1685	1016
2	1	1685	1016	10910	1749

Using the scorer nodes built in ability to calculate evaluation metrics, we generate the following metrics:



Recall ↓ Number (double)	Precision Number (double)	Sensitivity Number (double)	Specificity Number (double)	F-measure Number (double)	Accuracy Number (double)	Cohen's kappa Number (double)
0.915	0.862	0.915	0.491	0.888	?	?
0.491	0.624	0.491	0.915	0.549	?	?
?	?	?	?	?	0.82	0.439

Where the first row is 0, the second row is 1 and the third row is the overall scores. For simplicity, we will speak about each row separately then talk about the overall score cumulatively.

Row 1 (0) –

- Recall (Sensitivity): 0.915 - The model correctly identifies 91.5% of the non-rainy instances. This is a high recall, indicating that the model is effective at identifying days when it won't rain.
- Precision: 0.862 - When the model predicts "no rain," it is correct 86.2% of the time. This is also strong, showing that false positives are relatively low.

- Specificity: 0.491 - The model correctly identifies 49.1% of the actual rainy days as "rain," suggesting it misclassifies a considerable portion of rainy days as non-rainy. This low specificity indicates a bias towards predicting "no rain."
- F-measure: 0.888 - The F-measure (harmonic mean of precision and recall) is high, confirming good balance between precision and recall for the non-rainy class.

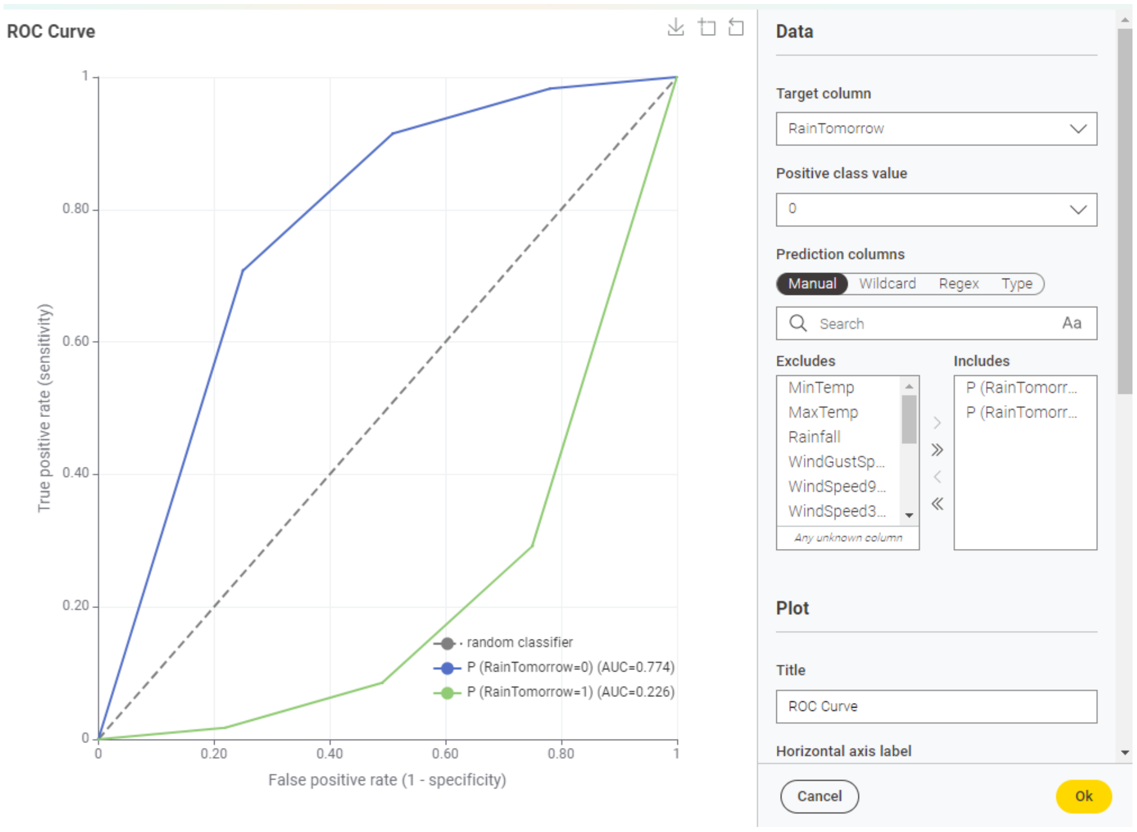
Row 2 (1) –

- Recall (Sensitivity): 0.491 - The model correctly identifies only 49.1% of actual rainy days, indicating low sensitivity for this class. This suggests a significant number of rainy days are misclassified as non-rainy.
- Precision: 0.624 - When the model predicts "rain," it is correct only 62.4% of the time. This moderate precision reflects some effectiveness but indicates that the model makes false positives for rain predictions.
- Specificity: 0.915 - The model correctly identifies 91.5% of non-rainy days as "no rain" when it predicts no rain, which is a strong indicator of how well it identifies the non-rainy days when considering the rain class.
- F-measure: 0.549 - This lower F-measure reflects the lower balance between precision and recall for the rain class, as the model is not as effective at identifying rain.

Overall performance –

- Accuracy: 0.82 - The model achieves 82% overall accuracy, which is reasonably good. However, because of the class imbalance and the model's bias towards the no-rain class, accuracy alone may not fully reflect the model's reliability.
- Cohen’s Kappa: 0.439 - This value indicates moderate agreement between the model predictions and actual values, accounting for chance. A Kappa of 0.439 suggests the model is better than random but does not reach a strong level of reliability.

ROC Curve 1:


ROC Curve
1
0.80
0.60
0.40
0.20
0
True positive rate (sensitivity)
0
0.20
0.40
0.60
0.80
1
False positive rate (1 - specificity)
random classifier
P (RainTomorrow=0) (AUC=0.774)
P (RainTomorrow=1) (AUC=0.226)

Data

Target column: RainTomorrow

Positive class value: 0

Prediction columns

Manual | Wildcard | Regex | Type

Search Aa

Excludes	Includes
MinTemp	P (RainTomorr...
MaxTemp	P (RainTomorr...
Rainfall	
WindGustSp...	
WindSpeed9...	
WindSpeed3...	
Any unknown column	

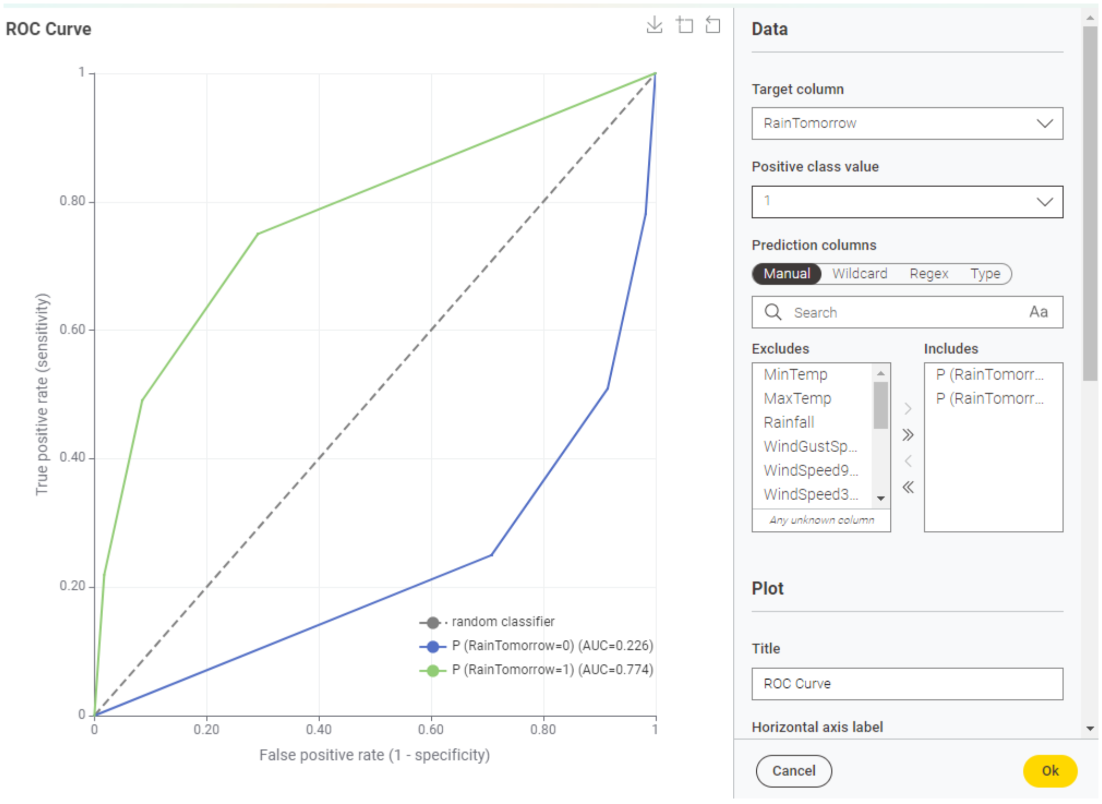
Plot

Title: ROC Curve

Horizontal axis label

Cancel | Ok

ROC Curve 2:


ROC Curve
1
0.80
0.60
0.40
0.20
0
True positive rate (sensitivity)
0
0.20
0.40
0.60
0.80
1
False positive rate (1 - specificity)
random classifier
P (RainTomorrow=0) (AUC=0.226)
P (RainTomorrow=1) (AUC=0.774)

Data

Target column: RainTomorrow

Positive class value: 1

Prediction columns

Manual | Wildcard | Regex | Type

Search Aa

Excludes	Includes
MinTemp	P (RainTomorr...
MaxTemp	P (RainTomorr...
Rainfall	
WindGustSp...	
WindSpeed9...	
WindSpeed3...	
Any unknown column	

Plot

Title: ROC Curve

Horizontal axis label

Cancel | Ok

The KNN classifier managed to achieve an accuracy of 0.82 on the testing data. This suggests that the performance of the model is good. The ROC curves suggest a class imbalance with non-rainy days being 0.774 and rainy days being 0.226. The issue arises with the minority classes being ungeneralised. This imbalance allowed for weaker results on rainy days and stronger results for the non-rainy days (majority). In essence, this means the model is biased for non-rainy days.

Analysis of the metrics revealed high precision and recall for non-rainy days, however, showed lower precision and recall for rainy days. This proves the bias of the classifier.

This result is extremely like the decision tree classifier. This means that both models handle the trade-offs between false positives and false negatives in a comparable way.

As the K Nearest Neighbour model does not generate a model, but instead, uses the inputted data to create clusters; there is no "Predictor" node like we had for the Decision Tree and Random Forest models. Although we do not have a predictor node, we are still able to use our model to find "RainTomorrow" values for our "UnknownData.csv". Using the KNN model, we managed to achieve a Kaggle score of 0.73080.

5.4 Model 4: Multi-Layer Perceptron (MLP)

A Multilayer Perceptron (MLP) is a type of artificial neural network (ANN) consisting of multiple layers of nodes, also known as "neurons." It is designed to approximate complex functions and is commonly used for supervised learning tasks, such as classification and regression. An MLP is called "multilayer" because it has at least three layers:

1. Input Layer: Receives the input features and passes them to the next layer without performing any computation.
2. Hidden Layer(s): One or more intermediate layers between the input and output layers, where the actual processing occurs. Each node in a hidden layer applies a weighted sum of its inputs followed by a non-linear activation function. These hidden layers allow the MLP to capture complex, non-linear relationships in the data.
3. Output Layer: Produces the final predictions or classifications. The activation function for this layer depends on the task; for example, a sigmoid function for binary classification or a softmax function for multiclass classification.

The key features of MLP are:

- Fully Connected: Each neuron in one layer is connected to every neuron in the next layer, enabling the network to learn complex patterns.

- Feedforward Structure: Information moves in one direction from the input layer, through the hidden layers, to the output layer.
- Backpropagation for Training: The model is trained using backpropagation, an optimization algorithm that adjusts weights to minimize the error between predicted and actual outcomes.

Multi-Layer Perceptron learner

The MLP Learner node is used to train a Multilayer Perceptron (MLP) neural network model for classification tasks. The RProp MLP Learner node specifically uses the Resilient Backpropagation (RProp) algorithm, which is an optimization method designed to improve the convergence speed by adapting the weight updates during training.

Shown below is the configuration used for my MLP learner node:

- Maximum number of iterations: Set to 100 in this configuration, this parameter specifies the maximum number of times the model will iterate over the data during training. Increasing this number could allow the model to learn more complex patterns but might also risk overfitting.
- Number of hidden layers: Set to 1, this option specifies the number of hidden layers in the neural network. In this case, there's only one hidden layer, making the network relatively simple. Adding more hidden layers can increase the model's ability to capture complex relationships but also increases the risk of overfitting and computational demands.

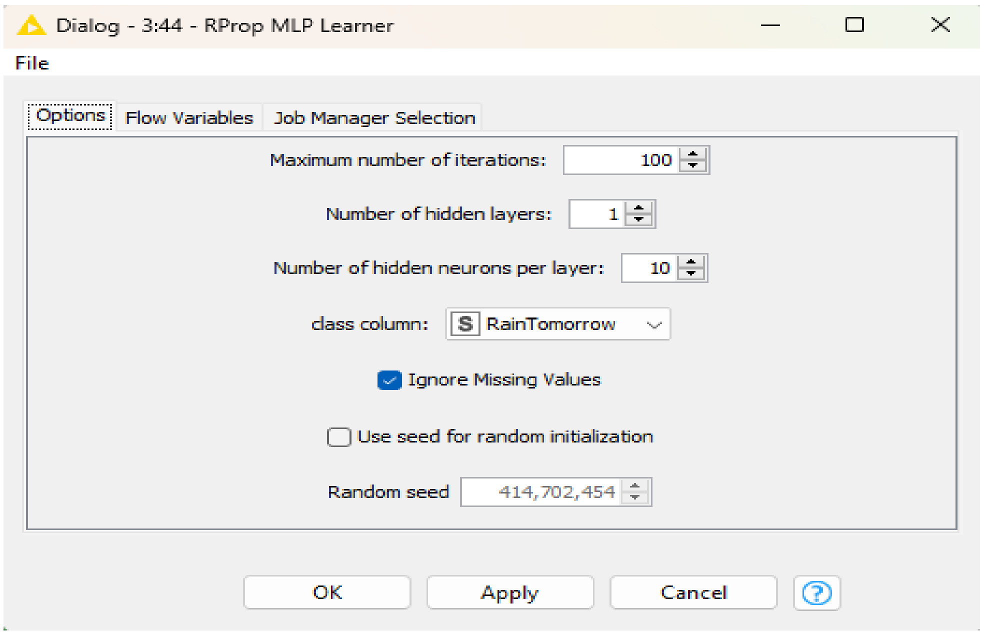
- Number of hidden neurons per layer: Set to 10, this parameter determines how many neurons are in each hidden layer. More neurons allow the model to learn more features from the data, but they also increase the complexity and computational cost of the model.

- Class column: Set to RainTomorrow, this is the target or dependent variable the model will predict. In this case, it's a binary classification task, predicting whether it will rain tomorrow.

- Ignore Missing Values: This option is checked, meaning the model will ignore rows with missing values during training. This can be useful to avoid complications from incomplete data, but if many rows have missing values, it could reduce the dataset size significantly.

- Use seed for random initialization: This option is unchecked. If enabled, it would allow for reproducibility by setting a fixed seed for the random initialization of weights. Without a seed, the training process will be randomized each time, leading to slightly different results on each run.

This information can be seen through this image of the node configuration screen:


Dialog - 3:44 - RProp MLP Learner

File

Options | Flow Variables | Job Manager Selection

Maximum number of iterations: 100

Number of hidden layers: 1

Number of hidden neurons per layer: 10

class column: S RainTomorrow

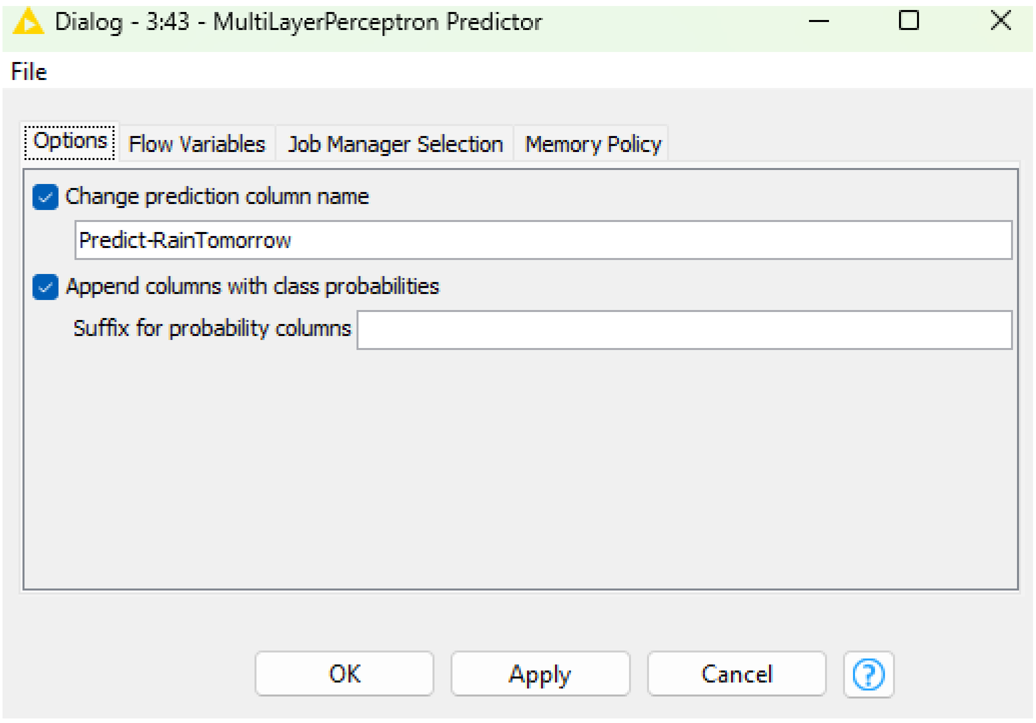
[✓] Ignore Missing Values

[ ] Use seed for random initialization

Random seed 414,702,454

Using the MultiLayerPerceptron Predictor node, we can use the model created from the learner node to generate values. The MLP Predictor node is used to make predictions on new data using a trained Multilayer Perceptron (MLP) model. After training an MLP model with the MLP Learner node, the MLP Predictor node applies that model to a test or validation dataset, generating predictions based on the patterns the model has learned.

Configuration of MultiLayerPerceptron Predictor node:


Dialog - 3:43 - MultiLayerPerceptron Predictor

File

Options | Flow Variables | Job Manager Selection | Memory Policy

[✓] Change prediction column name

Predict-RainTomorrow

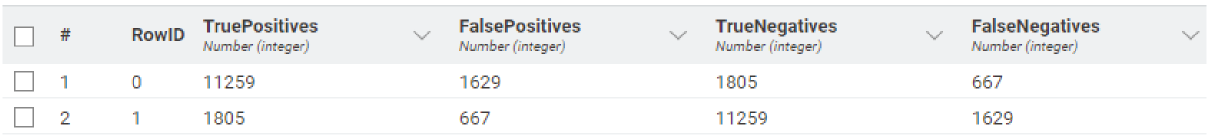
[✓] Append columns with class probabilities

Suffix for probability columns

Change prediction column name: This option is checked, and the field is set to Predict-RainTomorrow. This means the output column containing the predictions for the target variable will be named "Predict-RainTomorrow" instead of the default name. Renaming the prediction column can help keep results organized, especially if multiple models are being compared, or if the user wants to clearly distinguish between actual and predicted values. This was also done to fall in line with the requirements of the Kaggle submission challenge.

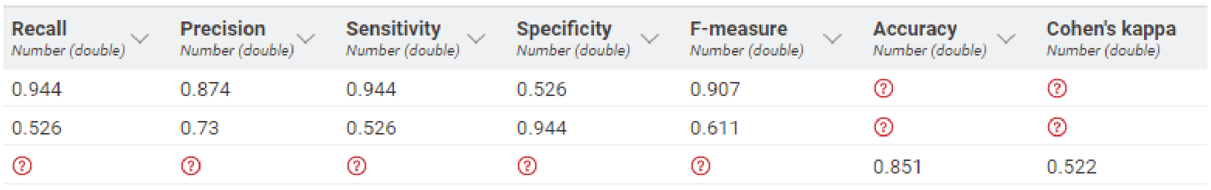
Append columns with class probabilities: This option is checked, which enables the node to add additional columns with the probability values for each class (e.g., the probability that it will rain tomorrow and the probability that it will not rain tomorrow). This is useful for tasks where understanding the certainty of predictions is important, as it provides more detailed insights into how confident the model is in each prediction.

Shown below is the confusion matrix for the MLP predictor which was gathered from the scorer node:



#	RowID	TruePositives Number (integer)	FalsePositives Number (integer)	TrueNegatives Number (integer)	FalseNegatives Number (integer)
1	0	11259	1629	1805	667
2	1	1805	667	11259	1629

Additionally, more information can be acquired from the scorer such as:



Recall Number (double)	Precision Number (double)	Sensitivity Number (double)	Specificity Number (double)	F-measure Number (double)	Accuracy Number (double)	Cohen's kappa Number (double)
0.944	0.874	0.944	0.526	0.907	?	?
0.526	0.73	0.526	0.944	0.611	?	?
?	?	?	?	?	0.851	0.522

For simplicity, a row-by-row analysis of the metrics will be completed.

No Rain (Row 1):

- Recall (0.944): The model correctly identifies 94.4% of the "no rain" cases, indicating strong performance in recognizing instances when it will not rain.
- Precision (0.874): When the model predicts "no rain," it is correct 87.4% of the time, suggesting some false positives but still a relatively high level of accuracy.
- Sensitivity (0.944): Same as recall, confirming the model's effectiveness at identifying true "no rain" instances.
- Specificity (0.526): This value is lower, indicating the model struggles to reject "no rain" predictions when it actually rains.
- F-measure (0.907): This combines precision and recall, showing a strong balance for "no rain" predictions.

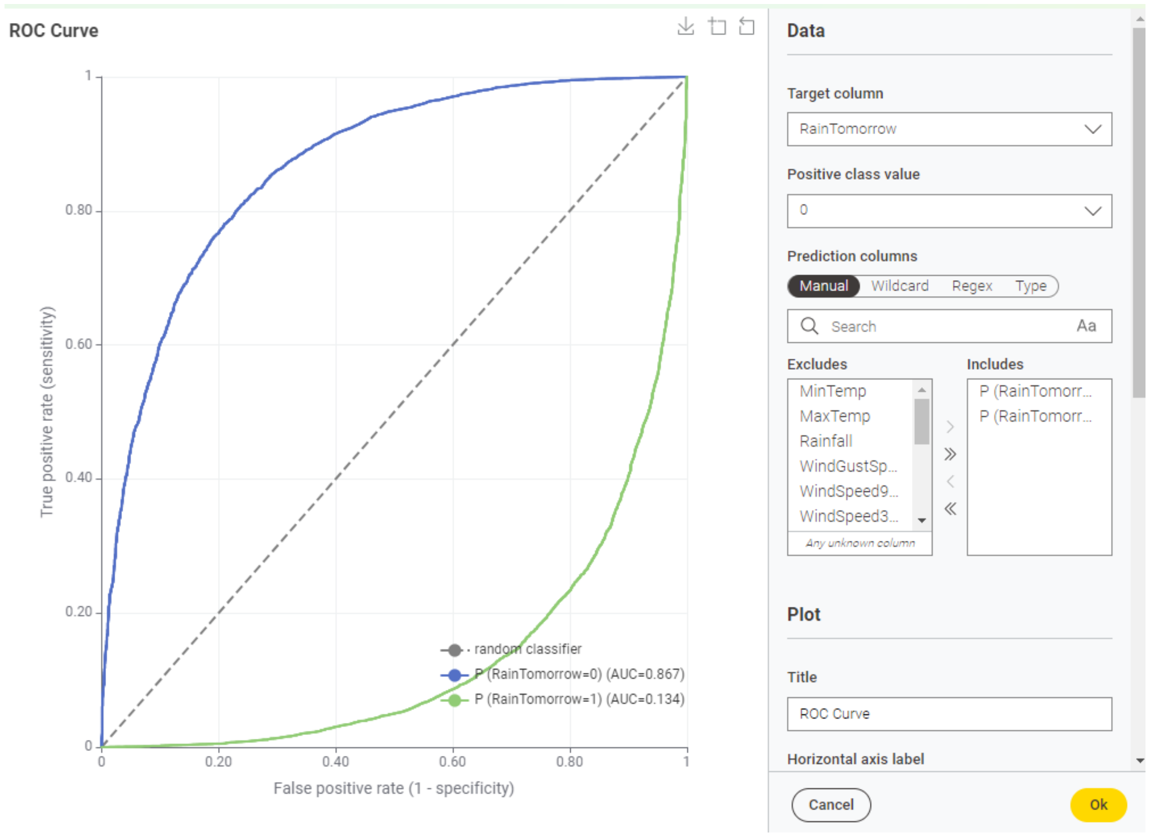
Rain (Row 2):

- Recall (0.526): The model correctly identifies only 52.6% of the "rain" cases, indicating it misses many instances when it should predict rain, potentially due to class imbalance.
- Precision (0.73): When the model predicts "rain," it is correct 73% of the time, indicating moderate accuracy for rain predictions.
- Sensitivity (0.526): This aligns with recall, further showing the model's limitation in detecting true "rain" cases.
- Specificity (0.944): The model correctly identifies "no rain" when it predicts "rain" at a high rate of 94.4%, suggesting it is more confident in predicting "no rain."
- F-measure (0.611): The balance between precision and recall is lower for rain, indicating weaker performance in accurately predicting rainy days.

Overall (Row 3):

- Accuracy (0.851): The model has an overall accuracy of 85.1%, meaning it correctly predicts the outcome (rain or no rain) about 85% of the time.
- Cohen's Kappa (0.522): This metric accounts for chance and suggests a moderate agreement between the model's predictions and the actual outcomes, highlighting room for improvement.

ROC Curve 1:


ROC Curve
1
0.80
0.60
0.40
0.20
0
True positive rate (sensitivity)
0
0.20
0.40
0.60
0.80
1
False positive rate (1 - specificity)
random classifier
P (RainTomorrow=0) (AUC=0.867)
P (RainTomorrow=1) (AUC=0.134)
Cancel
Ok

Data

Target column: RainTomorrow

Positive class value: 0

Prediction columns: Manual | Wildcard | Regex | Type

Search Aa

Excludes: MinTemp, MaxTemp, Rainfall, WindGustSp..., WindSpeed9..., WindSpeed3..., Any unknown column

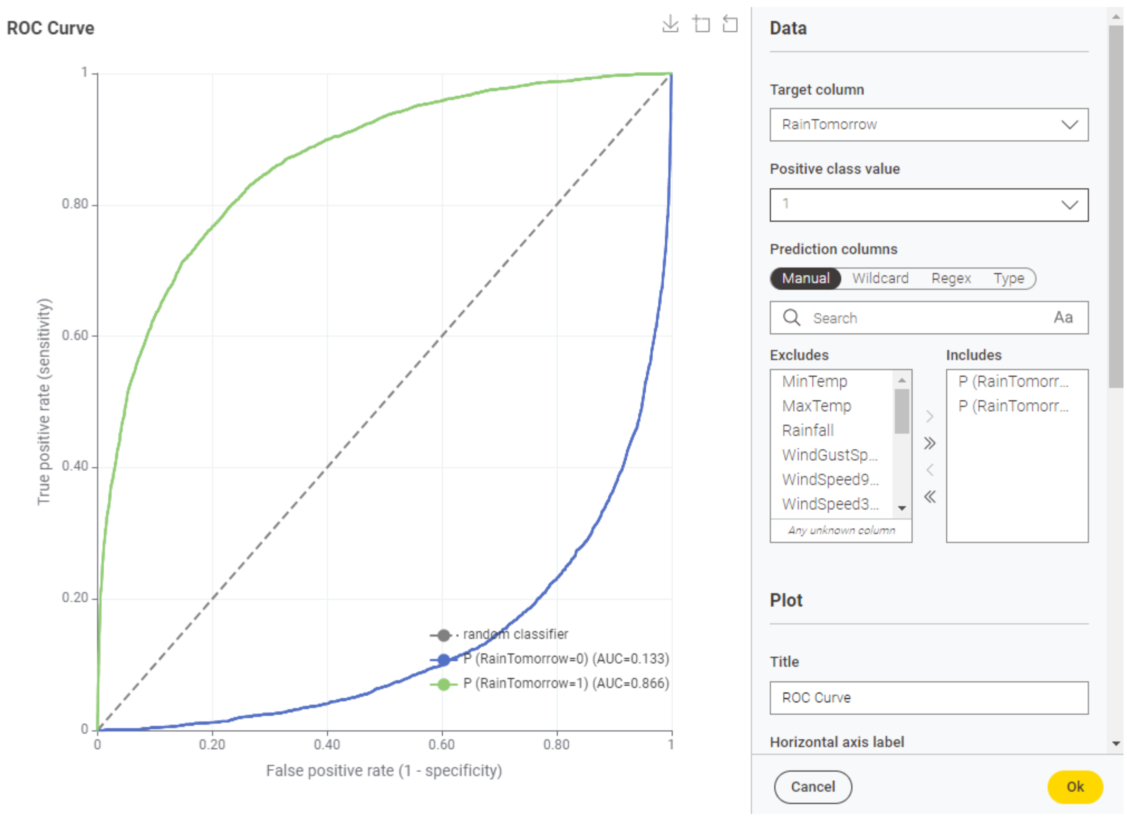
Includes: P (RainTomorr..., P (RainTomorr...

Plot

Title: ROC Curve

Horizontal axis label

ROC Curve 2:


ROC Curve
1
0.80
0.60
0.40
0.20
0
True positive rate (sensitivity)
0
0.20
0.40
0.60
0.80
1
False positive rate (1 - specificity)
random classifier
P (RainTomorrow=0) (AUC=0.133)
P (RainTomorrow=1) (AUC=0.866)
Cancel
Ok

Data

Target column: RainTomorrow

Positive class value: 1

Prediction columns: Manual | Wildcard | Regex | Type

Search Aa

Excludes: MinTemp, MaxTemp, Rainfall, WindGustSp..., WindSpeed9..., WindSpeed3..., Any unknown column

Includes: P (RainTomorr..., P (RainTomorr...

Plot

Title: ROC Curve

Horizontal axis label

Our ROC curves indicate that the model is optimised for predicting "no rain" (0) as the positive class. When "no rain" is set as positive, it achieves a high AUC of 0.867, showing strong predictive performance for non-rainy days. However, when "rain" is set as positive, the AUC remains high for rain but low for "no rain," suggesting that the model is not as effective in predicting rainy days accurately.

This discrepancy suggests that the model may be biased toward the majority class (no rain), performing well when "no rain" is the positive class but struggling with "rain" as the positive class, likely due to class imbalance. Additional balancing techniques, like further oversampling of the minority class or tweaking the model’s decision threshold, could help improve the model’s ability to predict rainy days accurately. This model managed to achieve a Kaggle score of 0.75507.

5.5 Model 5: Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised machine learning algorithm commonly used for classification tasks, although it can also be adapted for regression. SVMs are known for their effectiveness in high-dimensional spaces and their ability to handle both linearly and non-linearly separable data.

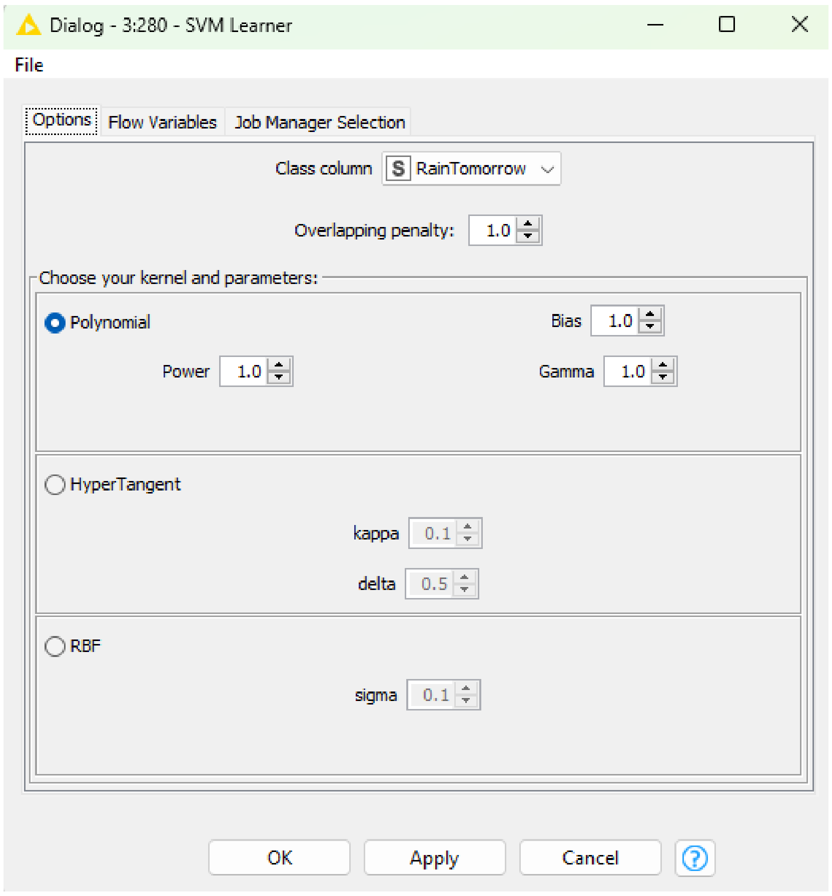
There are two types of SVM’s:

- Linear SVM: Used when the data is linearly separable. It finds a straight hyperplane that maximizes the margin.
- Non-linear SVM: Used when the data is not linearly separable. Here, the kernel trick is applied to map data into a higher-dimensional space, enabling linear separation.

SVM learner

The SVM Learner node is used to train a Support Vector Machine (SVM) model on a labelled dataset. It is part of KNIME's machine learning suite and can handle both classification and regression tasks, though it's primarily used for classification. This node allows users to configure and train an SVM model by specifying parameters such as the kernel type, regularization, and stopping criteria.

Configuration of the SVM Learner node:


Dialog - 3:280 - SVM Learner

File

Options | Flow Variables | Job Manager Selection

Class column: RainTomorrow

Overlapping penalty: 1.0

Choose your kernel and parameters:

[✓ Polynomial] — Bias: 1.0, Power: 1.0, Gamma: 1.0

HyperTangent — kappa: 0.1, delta: 0.5

RBF — sigma: 0.1

As before, we will be talking about the configuration from top down. Starting with:

- Class Column: Set to RainTomorrow

- This specifies the target column that the SVM model will be trained to predict. In this case, RainTomorrow is the variable we want to classify.
- Overlapping Penalty (C parameter): Set to 1.0

- This parameter controls the trade-off between maximizing the margin and minimizing classification errors. A higher value places a heavier penalty on misclassifications, which can lead to a narrower margin. A lower value allows for more flexibility, potentially increasing the margin but risking some misclassifications. Here, it is set to 1.0, which represents a balanced approach.

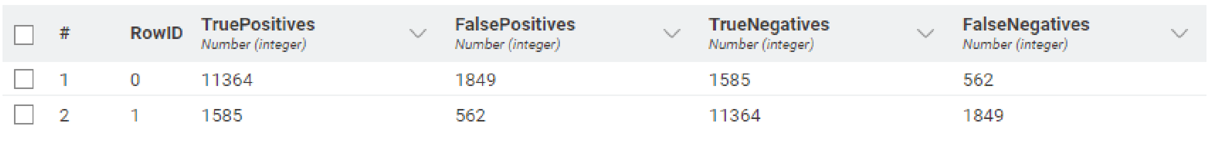
- Kernel Choice: Polynomial
  - This configuration has selected the Polynomial kernel, which is useful for non-linear relationships. The polynomial kernel projects the data into a higher-dimensional space, enabling the model to find non-linear decision boundaries. Polynomial kernels are effective when data cannot be

separated by a straight line but has a pattern that a polynomial curve could capture.

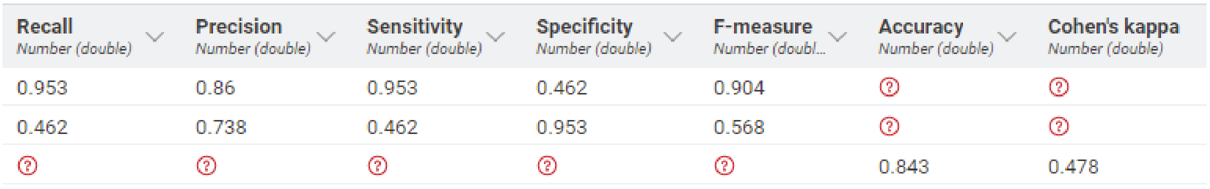
- Polynomial Kernel Parameters:
    - Bias: Set to 1.0
        - The bias term shifts the decision boundary away from the origin, helping the model fit the data more flexibly. A higher bias can increase the model's flexibility in finding decision boundaries for complex data.
    - Power: Set to 1.0
        - This is the degree of the polynomial. A higher power means the polynomial kernel will create a more complex decision boundary, which may help with more complicated data structures but could also lead to overfitting.
    - Gamma: Set to 1.0
        - The gamma parameter defines the influence of each training sample. A high gamma value means the model will focus more closely on individual points, which can lead to overfitting. A lower gamma value makes the model generalize more by considering the broader structure of the data. Here, gamma is set to 1.0, balancing between specificity and generalization.

In truth, the configuration for this node has been kept identical to the default settings. The reason for this is because, after tampering, the other kernel choices did not make sense in regard to our desired outcome. Additionally, changing the bias, power, and gamma values from 1.0, would decrease the accuracy of the model. Thus, the configuration has been kept as default.

Using the scorer node, we can see exactly what I am talking about. The confusion matrix and evaluation metrics are shown below:



#	RowID	TruePositives Number (integer)	FalsePositives Number (integer)	TrueNegatives Number (integer)	FalseNegatives Number (integer)
1	0	11364	1849	1585	562
2	1	1585	562	11364	1849



Recall Number (double)	Precision Number (double)	Sensitivity Number (double)	Specificity Number (double)	F-measure Number (doubl...	Accuracy Number (double)	Cohen's kappa Number (double)
0.953	0.86	0.953	0.462	0.904	?	?
0.462	0.738	0.462	0.953	0.568	?	?
?	?	?	?	?	0.843	0.478

Like MLP, we will be completing a row-by-row analysis of the evaluation metrics, where:

The first row is 0 (No rain)

The second row is 1 (Rain)

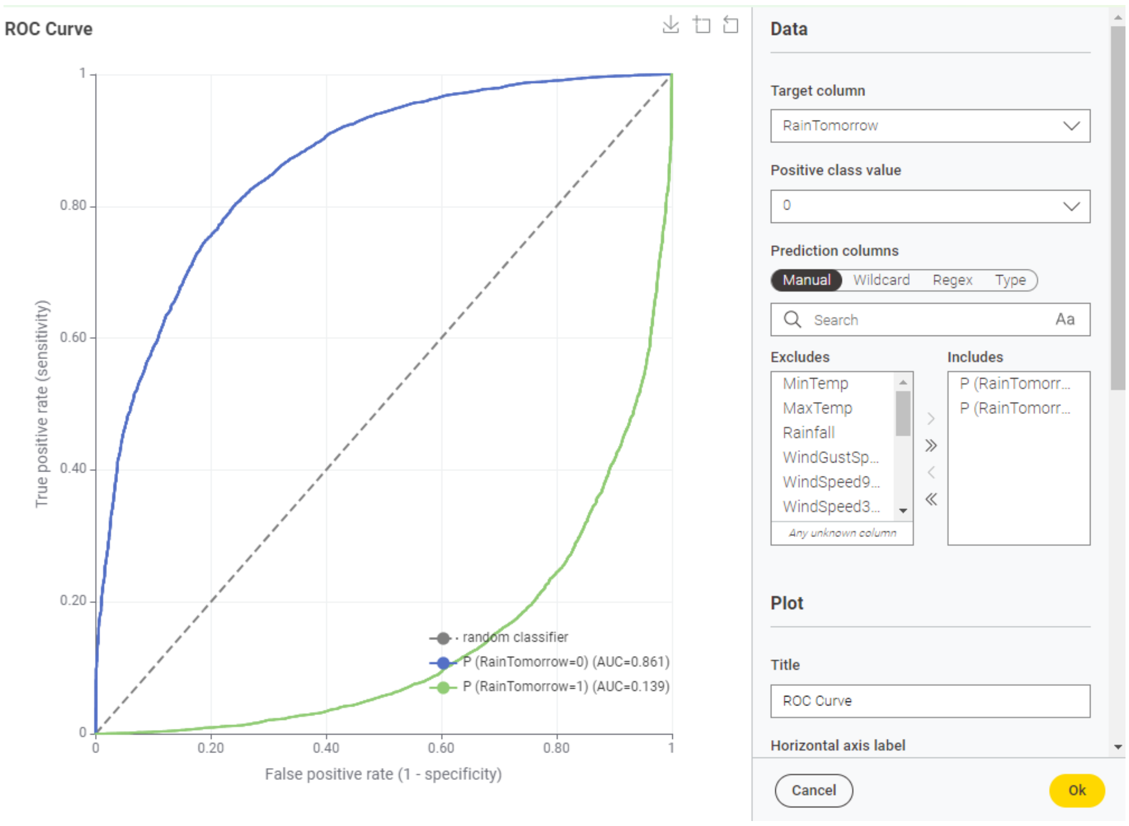
The third row is Overall performance

- No Rain (Row 1):
    - Recall (0.953): The model correctly identifies 95.3% of the "no rain" cases, showing high effectiveness in recognizing instances when it will not rain.
    - Precision (0.86): When the model predicts "no rain," it is correct 86% of the time, indicating a relatively high level of accuracy with some false positives.
    - Sensitivity (0.953): This matches recall, reaffirming the model's strong performance in detecting "no rain" instances.
    - Specificity (0.462): This lower specificity suggests that the model struggles to reject "no rain" predictions when it should predict rain, which may lead to some misclassifications of rainy days as non-rainy.
    - F-measure (0.904): This combines precision and recall, showing a strong balance in "no rain" predictions, indicating that the model performs well in predicting no rain.

- Rain (Row 2):
    - Recall (0.462): The model correctly identifies only 46.2% of the "rain" cases, indicating it misses many instances when it should predict rain, likely due to an imbalance in the data.
    - Precision (0.738): When the model predicts "rain," it is correct 73.8% of the time, showing moderate accuracy for rain predictions but still a notable number of false positives.
    - Sensitivity (0.462): This is the same as recall, further highlighting the model's limitation in detecting true "rain" cases.
    - Specificity (0.953): The model is highly specific in identifying "no rain" when it predicts "rain," with a 95.3% rate of correctly rejecting "rain" predictions when they are actually "no rain."
    - F-measure (0.568): The balance between precision and recall for rain is lower than for no rain, indicating weaker performance in accurately predicting rainy days.

- Overall (Row 3):
    - Accuracy (0.843): The model has an overall accuracy of 84.3%, meaning it correctly predicts the outcome (rain or no rain) about 84% of the time, indicating decent performance overall.
    - Cohen's Kappa (0.478): This metric accounts for chance and indicates moderate agreement between the model's predictions and the actual outcomes, suggesting that the model could benefit from improvements, especially for the minority class.

ROC Curve 1:


ROC Curve
1
0.80
0.60
0.40
0.20
0
True positive rate (sensitivity)
0
0.20
0.40
0.60
0.80
1
False positive rate (1 - specificity)
random classifier
P (RainTomorrow=0) (AUC=0.861)
P (RainTomorrow=1) (AUC=0.139)

Data

Target column: RainTomorrow

Positive class value: 0

Prediction columns: Manual | Wildcard | Regex | Type

Search Aa

Excludes: MinTemp, MaxTemp, Rainfall, WindGustSp..., WindSpeed9..., WindSpeed3..., Any unknown column

Includes: P (RainTomorr..., P (RainTomorr...

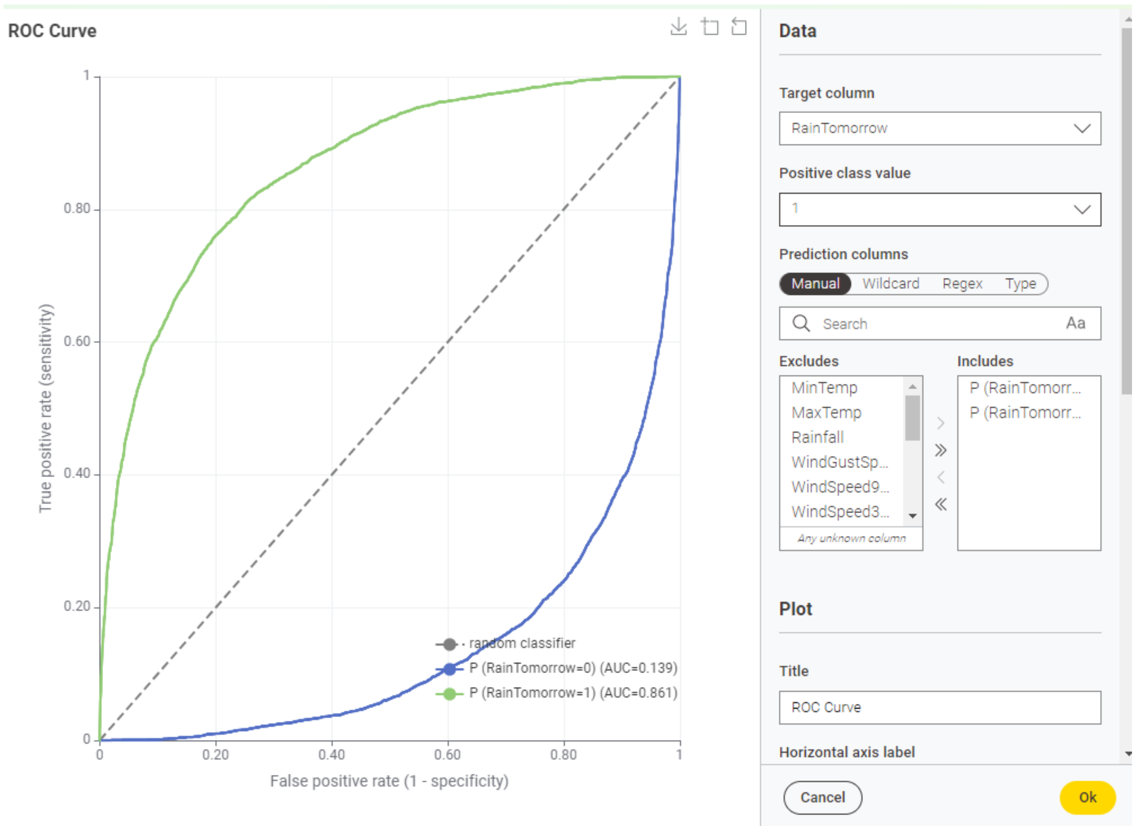
Plot

Title: ROC Curve

Horizontal axis label

Cancel Ok

Roc Curve 2:


ROC Curve
1
0.80
0.60
0.40
0.20
0
True positive rate (sensitivity)
0
0.20
0.40
0.60
0.80
1
False positive rate (1 - specificity)
random classifier
P (RainTomorrow=0) (AUC=0.139)
P (RainTomorrow=1) (AUC=0.861)

Data

Target column: RainTomorrow

Positive class value: 1

Prediction columns: Manual | Wildcard | Regex | Type

Search Aa

Excludes: MinTemp, MaxTemp, Rainfall, WindGustSp..., WindSpeed9..., WindSpeed3..., Any unknown column

Includes: P (RainTomorr..., P (RainTomorr...

Plot

Title: ROC Curve

Horizontal axis label

Cancel Ok

The ROC curves indicate that the model is optimized for predicting the dominant class, which might be "no rain," given the high AUC of 0.861 in both cases when either rain or no rain is set as the positive class. However, the model performs poorly in identifying the minority class, which is reflected in the low AUC of 0.139 when the non-prioritized class (either rain or no rain) is set as the negative class. This suggests a class imbalance or an inherent model bias toward the majority class.

To improve the model’s ability to predict the minority class, techniques like further oversampling of the minority class, tweaking the decision boundary, or using alternative kernels or regularization parameters might help balance the model’s performance across both classes.

In summary, every model seemed to perform very similarly, as such, each model faced the same issue. Each model was exceptional at predicting “No rain” but struggled to predict the “Rain” values. This was because of bias being present in the dataset for the majority class which in our case is the “No rain” value. As stated, prior, the best solution to all of these issues is to oversample the minority classes further.

	Scorer node Accuracy score	Kaggle Submission score
Decision Tree	0.833	0.71281
Random Forest	0.848	0.75429
K Nearest Neighbour	0.82	0.73080
MLP	0.851	0.75507
SVM	0.843	0.71891

According to our table, the best performing model was the Multi-Layer Perceptron model. With the highest scores achieved in both the accuracy score and the highest score in the Kaggle submission. For this reason, I am selecting this model as the best performing model.

MLP solves a classification problem by learning to map input features to output classes through a network of interconnected neurons, organized in layers. To understand this better, here is a step-by-step layout of how an MLP model is trained and used for prediction to solve our problem:

1. Input Layer:

- The input layer receives the features from the dataset. Each input feature (e.g., temperature, humidity) corresponds to a neuron in the input layer.
- These inputs are normalized or scaled to ensure consistent ranges, as this helps improve learning stability and performance.

2. Hidden Layers and Neurons:

- The input features are passed through one or more hidden layers, which are the core computational layers of the MLP.
- Each neuron in a hidden layer calculates a weighted sum of its inputs, applies an activation function, and passes the result to the next layer.
- The activation function introduces non-linearity, enabling the MLP to capture complex relationships in the data that wouldn't be possible with just linear transformations.

3. Output Layer:

- For a classification problem, the output layer usually consists of one neuron per class. For binary classification, there is typically a single neuron that outputs a probability of belonging to a specific class.
- The output layer applies an activation function suited for classification, such as:
    - Sigmoid: For binary classification, giving a probability output between 0 and 1.
    - Softmax: For multiclass classification, giving a probability distribution across multiple classes.
- The output is interpreted as the predicted class (the one with the highest probability) for each input.

4. Training with Backpropagation:

- The MLP is trained using backpropagation, an algorithm that adjusts the weights of the neurons to minimize the prediction error.
- During training, the model makes predictions, compares them to the actual class labels, and calculates a loss (error) using a loss function, such as cross-entropy for classification tasks.
- The error is then propagated back through the network, and weights are updated to reduce the loss in future predictions. This optimization process is often done with an algorithm like stochastic gradient descent (SGD) or one of its variants (e.g., Adam).

5. Iterative Learning:

- The training process involves multiple iterations, or epochs, during which the model adjusts its weights repeatedly based on the training data.
- Over time, the MLP learns patterns in the data that help it generalize well to unseen data, effectively mapping input features to the correct output class.

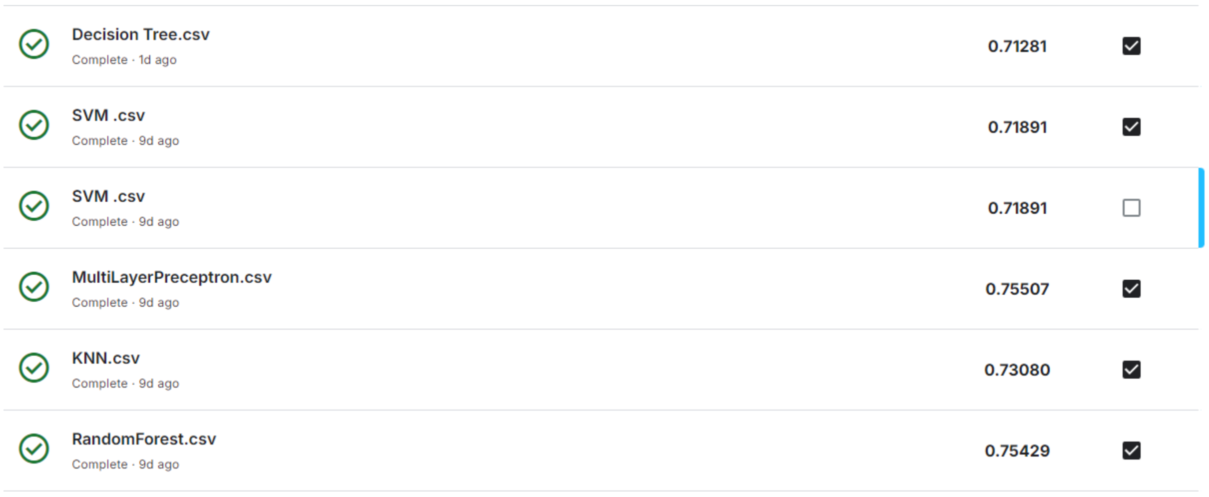
6. Making Predictions:

- After training, the MLP can classify new instances by processing them through the network layers.
- For each input, the MLP calculates the probabilities for each class and assigns the instance to the class with the highest probability.

6.0 Model Comparison & Final Selection

Based on the recorded performance prior, the MLP model trained using KNIME has successfully managed to achieve a high score in predicting "0" (No rain), however, struggled in predicting the "1" (Rain") This is like all the other models we have tested. However, the MLP model was selected as the best model, for not only achieving the highest scores in both Kaggle and accuracy, but also because of the efficiency of the model. Unlike models like SVM, the MLP model is very time efficient, this means that for extremely large datasets, the model can run much faster.

Kaggle submission score:



Decision Tree.csv Complete · 1d ago	0.71281	☑
SVM .csv Complete · 9d ago	0.71891	☑
SVM .csv Complete · 9d ago	0.71891	☐
MultiLayerPreceptron.csv Complete · 9d ago	0.75507	☑
KNN.csv Complete · 9d ago	0.73080	☑
RandomForest.csv Complete · 9d ago	0.75429	☑

The ticked boxes are the final submission score for the respective models.

7.0 Conclusion & Future Work

This project successfully navigated a complex, real-world dataset to build a functional rainfall prediction model. Through a rigorous process of exploratory data analysis, multi-stage preprocessing, and comparative model evaluation, a Multi-Layer Perceptron was identified as the most effective classifier, achieving 85.1% accuracy.

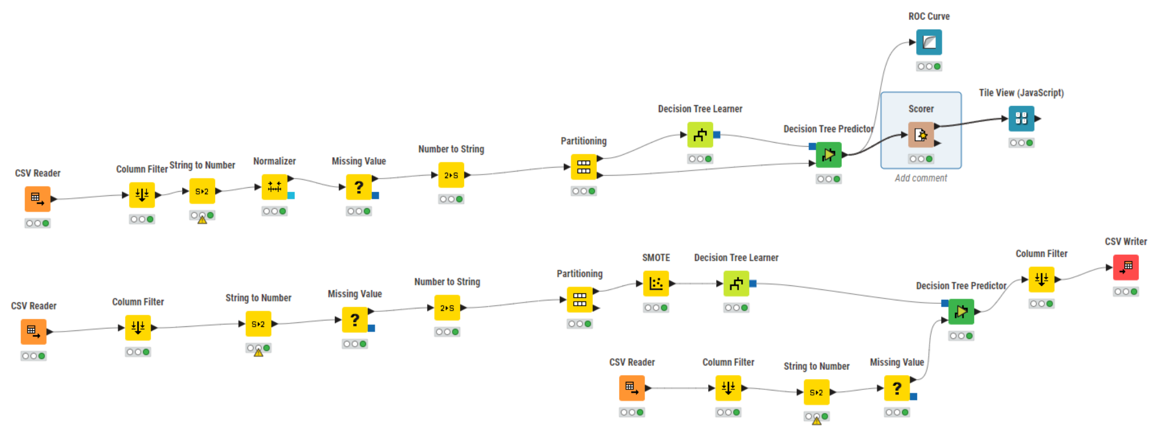
The primary challenge identified was the significant class imbalance, which led all models to develop a strong bias towards predicting the majority "No Rain" class. While techniques like SMOTE were used to mitigate this, further work is needed to improve the model's sensitivity to the minority "Rain" class.

Future work should include:

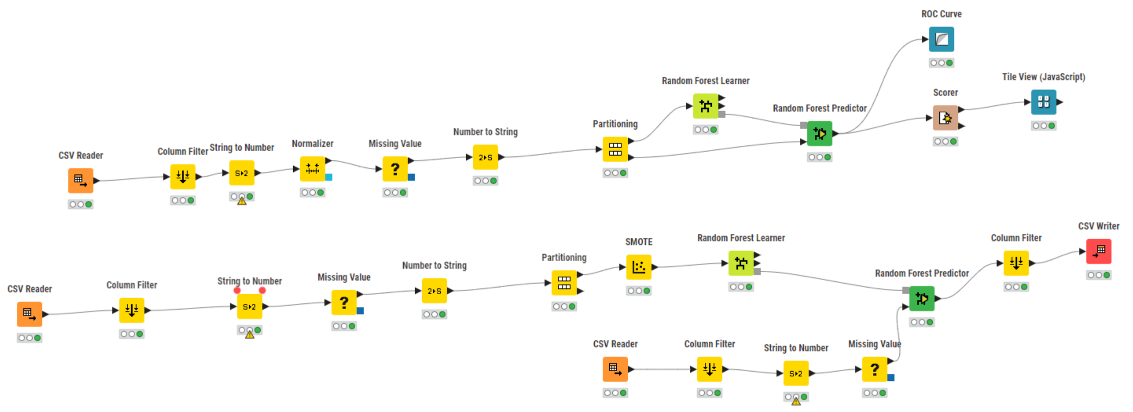
- **Advanced Feature Engineering:** Creating new features from existing data (e.g., interaction terms between temperature and humidity) could provide the models with stronger predictive signals.
- **Alternative Sampling Techniques:** Exploring more advanced over-sampling or under-sampling techniques beyond SMOTE may yield a better-balanced training set.
- **Ensemble Modeling:** Creating a "stacked" ensemble, where the predictions of multiple models are used as inputs for a final meta-model, could further improve predictive accuracy.

Appendix

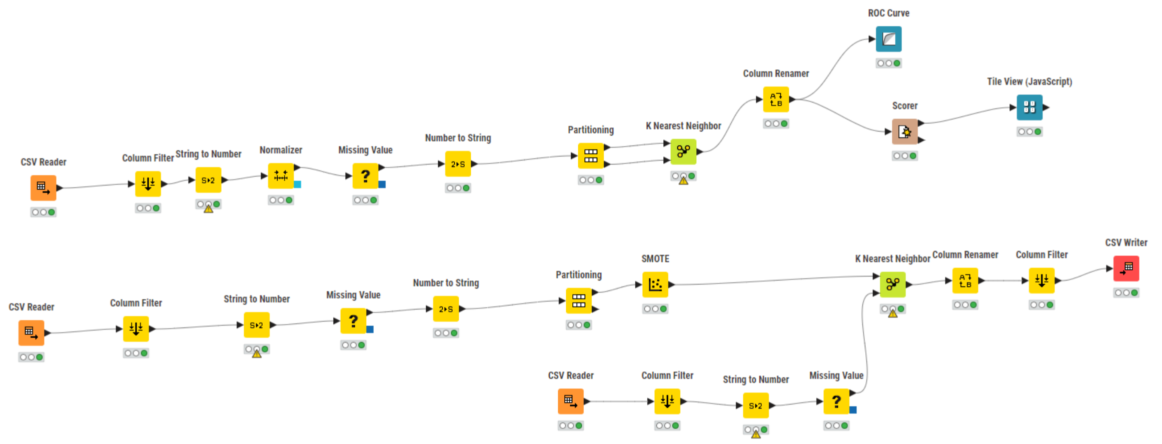
Decision Tree:



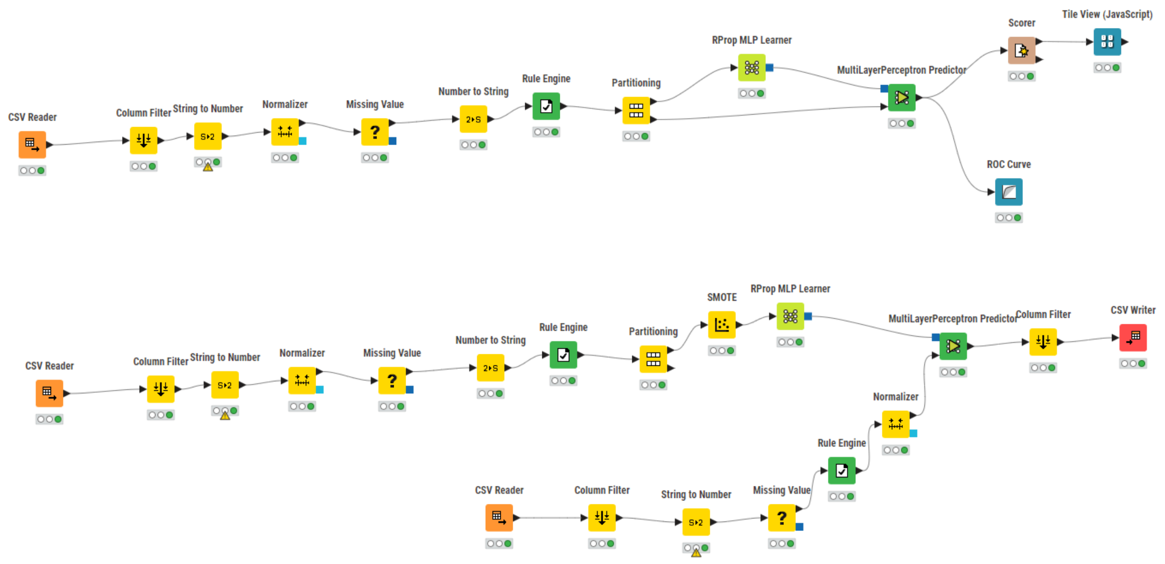
Random Forest:



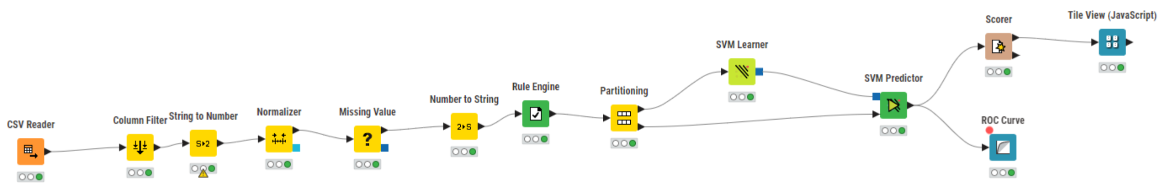
K Nearest Neighbour:

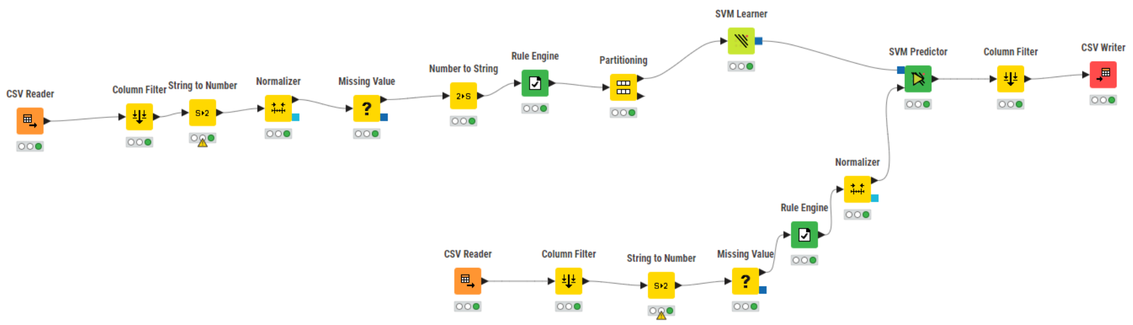


Multi-Layer Perceptron (MLP):



Support Vector Machine (SVM):


CSV Reader
Column Filter
String to Number
Normalizer
Missing Value
Number to String
Rule Engine
Partitioning
SVM Learner
SVM Predictor
Scorer
Tile View (JavaScript)
ROC Curve


CSV Reader
Column Filter
String to Number
Normalizer
Missing Value
Number to String
Rule Engine
Partitioning
SVM Learner
SVM Predictor
Column Filter
CSV Writer
CSV Reader
Column Filter
String to Number
Missing Value
Rule Engine
Normalizer