Machine Learning Model 🡪 Random Forest Classifier (RFC)

Is there a clear distinction between industrial and developing countries based on GHG emissions per year? Is one group contributing distinctly more GHG emissions? Are developing countries at a disadvantage for GHG emissions production or is there no distinction between the types of countries. Questions and thoughts along this framework lead us to creating a machine learning model using random forest classification to predict weather a country was industrial or developing based on equal features. Narrowing down on this classification would improve understanding of the production of GHG emissions.

From our dataset using specific features to train the RFC can it accurately select industrial or developing countries? Our target for the ML model was set early on. Our first RFC model was specifically designed to look at one year of data (2018), the latest data available. Then the code was modified to look at any year in the data set 1990 – 2018. Our latest RFC model considers all the years available.

The initial step required connecting the Jupyter Notebook to the SQL server that was hosted on AWS. The SQL database housed a verity of tables created with data sorted uniquely between tables. The table selected for the RFC had null values removed, which represented a small portion of the dataset, mostly from early years within the data set. This clean table allowed for solid data set to begin working with.

Preliminary data analysis:

* Identify features that distinguish developing countries from industrial countries.
  + Types of GHGs produced along with food system stages within countries.

Preparing the data for the machine learning model:

* Remove redundant columns from data such as country code.
* Remove international aviation and international shipping GHG emissions data.
* The target for the ML RFC is to distinguish countries from each other.
* Remove country grouping, this initially seemed redundant to include country names and group.
* Further analysis had interesting results.
* Encode each column using LabelEncoder to fit transform names to unique values for the machine learning model.
* Adjust the dataset to contain only columns used by machine learning model

Prepare for machine learning

* Split the working data frame into features and target values.
* Split the working data frame into a training and testing dataset.
* Scale and fit the data for consistency along the distribution of data points for the *RandomForestClassifier*

Advantages for the RFC model:

* RFC algorithm is suited for classification tasks.
* Consisting of many individual decision trees the RFC model prevents overfitting.
* Trees are trained on different pieces of the data that operate as an ensemble.
* Effective for processing many input variables when working with large datasets.

Limitations for the RFC model:

* Can require a significant amount computational resources to process the data.
* Can be time consuming for large data sets to train multiple trees.
* RFC can’t extrapolate, it can only make a prediction that are based previous observations.

Setting up the RFC model:

The target was defined as 0 for Developing and 1 for Industrial. The data frame for training the RFC model included the 4 types of GHG tracked in our data set along with stage of food system that corresponds to that specific GHG. To further help identify the target, the name of the country was also provided with the eighteen years of tracked data.

The rest of the data was split using the standard 75% training 25% testing ratio, this benchmark is a common and allows for training without overfitting, while leaving a robust data set for training.

The next step applied the StandardScaler to the training data set and fit the data.

Next the RFC was created via RandomForestClassifier() using two parameters, 1) the number of trees created by the algorithm (n\_estimators=500) and 2) a random\_state=1.

* N\_estimators = the number of trees to be built before taking the maximum voting or averages of predictions. The higher generally better accuracy but more CPU resources required.
* random\_state= set to check and validate the data when running the code multiple times, a fixed value will assure that the same sequence of random numbers is generated each time the code is run.

With the RFC built the model was fit the trained and scaled data set allowing for predictions.

After making predictions on the scaled testing data analysis was made on the Accuracy, Precision, Recall, and F1-Scores.

Interpreting the RFC model:

When our initial model analyzed specifically 2018, we had an overall accuracy score within a tenth compared to looking at all years (79% vs 78%). This was a similar trend observed running the model for various individual years in the data set (1995 Below). We typically observe the model able to predict developing countries near 80% accuracy with industrial countries near 70%. A quick summary of these scores tells us that industrial countries don’t have many unique features within the group to help the model identify them with the data provided. Further analysis could include additional futures such as population to help the model. A keynote to consider, when the model was tweaked to account for country grouping the RFC accuracy jumped to suspicious 98%. When the datasets country grouping was visualized, we see clear geographic boundaries and good reason why the accuracy jumps. Coupling the country grouping with the other features clearly boost the model’s accuracy. For example, if a country is within the Central Europe group it is 100% industrial. For our model without country grouping, country name was much more important than the model with country grouping. Finding those correlations was helpful to understand importance of different features and help model future work.

2018:

Table

Description automatically generated

1995:

Table

Description automatically generated

All Years with country grouping:

Table

Description automatically generated

All years without country grouping:

Table

Description automatically generated