# GEOG5927M: Predictive Analytics

Practical 3: Predicting travel to Palma de Mallorca; a random forest approach

# Aim

The aim of this week’s practical is to build a random forest algorithm to predict travel to Palma de Mallorca using a synthetic dataset. The data will be familiar to you from the previous practical. You will train and test a random forest algorithm, examine model metrics (i.e. how successful the model is) and identify highlight predictive variables which would prove useful in a targeting marketing strategy.

# Random Forest

Random Forest is a supervised machine learning method. It uses labelled training data to learn an algorithm which can then be used to label ‘unseen’ data, known as a testing set. Random Forest is made up of individual decision trees. The algorithm is often used for prediction purposes. In this practical we will attempt to use Random Forest to predict who might travel to Palma de Mallorca based upon their socio-demographic characteristics.

# Load the script and revisit the data

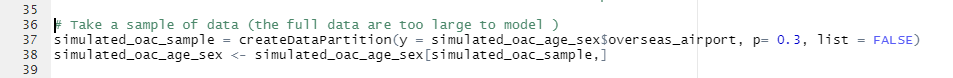
* 1. Open RStudio (via Appsanywhere, or local install). Create a new project (File > New Project, called ‘Practical\_3’. You have been provided with a script called ‘prac\_3\_RF. R’. You have also been provided with a workspace (in case the random forest model takes too long to train).
  2. The first task is to install and load the appropriate libraries. This practical mainly uses the Caret Library in R (*Classification and Regression Training*), developed by Max Kuhn. The documentation can be found here: <https://topepo.github.io/caret/>
* Caret contains many functions to streamline the creation of predictive models, including data splitting, pre-processing, model training, tuning and variable selection.
  1. We will now load in the individual data and simulated data from the previous practical and join together using inner\_join(). glimpse() is used to preview the data:

Text

Description automatically generated

# Pre-process the data

* 1. Ideally we would keep the entire dataset to train the algorithm, however the training process can take a long time, so for this practical we will use approximately a third of the data.
  2. We use the createDataPartition() command to sample the dataset. Note we pass in the overseas\_airport variable as a parameter, this ensures the class proportions remain intact.



* 1. Next we aggregate some of the classes for the household\_income variable. During the model training, categorical variables are converted to dummy variables (a column is created for each class outcome, where a value of ‘1’ indicates presence of that particular class outcome and ‘0’ indicates absence). Reducing the number of class outcomes can speed up the training process.

Text, letter

Description automatically generated

* 1. We will now subset the data to include only a limited number of predictor variables: oac\_group, sex, age\_band, number\_children and income\_band. We also include overseas\_airport (which our outcome variable is part of).

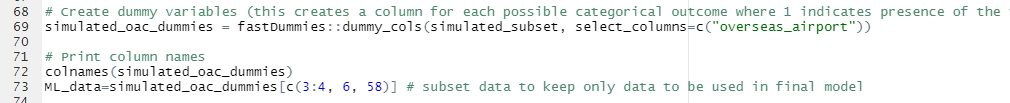


* 1. As we wish to undertake a classification (i.e. predict if someone is likely to visit Palma de Mallorca based on their socio-demographic characteristics), we need to convert categorical variables into factors. We use the as.factor() command to convert variables in simulated\_subset:

Text

Description automatically generated

* 1. Using the fastDummies library (<https://cran.r-project.org/web/packages/fastDummies/fastDummies.pdf>) We now manually create dummy variables for each class in overseas\_airport. This allows us to explicitly pass Palma to the model as out outcome class. I.e. we want to predict whether someone travelled to Palma (1) or not (0). We could have manually created a new column and populated using an if, else statement but the dummy\_cols() function is useful for creating multiple dummy columns at once. We only need the overseas\_airport\_PMI column, so the others are removed by subsetting.



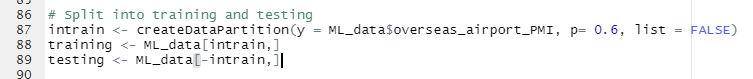
* 1. The final processing step is to check that the data is complete (i.e. there are no missing values). The complete.cases() command is used to do this and the outcome variable (overseas\_airport\_PMI) is concerted to a factor:

Text

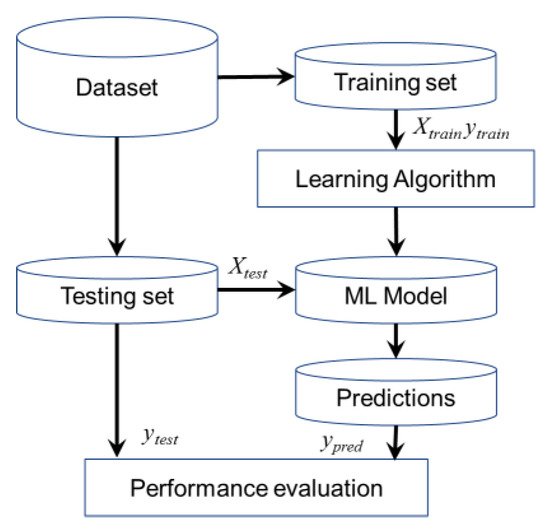
Description automatically generated

# Partition the data

* 1. Our data needs to be partitioned into testing and training data. The training data is typically comprised of 70% of the data, and it is used to create the model algorithm. The remaining 30% of the data is set aside to test our model – i.e. how good would it be at classifying unseen data? This ratio strikes a balance between ensuring enough data is available during the training process, and ensuring model metrics are robust. Again we use the createDataPartition() command from the caret package to split our data. As before, we pass in overseas\_airport\_PMI to ensure class proportions are maintained.



An overview of the training and testing process is outlined in the figure below (Chen et al. 2019):



Chen, Y.-T.; Piedad, E., Jr.; Kuo, C.-C. Energy Consumption Load Forecasting Using a Level-Based Random Forest Classifier. Symmetry **2019**, 11, 956. <https://doi.org/10.3390/sym11080956>

# Train and Test the Model

* 1. 10-fold cross-validation is used to train the model. This splits the training data into 10 equal folds, 9 are used for training the algorithm and the last fold is used to determine the effectiveness of the model. 10 is the default number of folds (note we do not explicitly pass a number into the train command). It is common to repeat the cross-validation process (repeatedcv), however this process takes time which we don’t have in this practical (see the end of the script for the repeated cv code).
  2. Immediately before we train the model we set the seed to a specific number, this is so we can reproduce the exact results when we run the model again. We also add some code to vary the ‘mtry’ variable. This is the number of randomly selected predictor variables to be considered at each split point (i.e. not all predictor variables are used in each decision tree). Firstly, mtry is set to the square root of the number of columns in the training set and passed to the tunegrid command for use in the training process. NOTE: the training process will take a while (upto 10 minutes), so be patient. If you are waiting longer than that, load in the R workspace provided which includes the trained models.

Text

Description automatically generated

* 1. We can print the model name (PMI\_tree) to see a summary:

Text, letter

Description automatically generated

* 1. To apply the algorithm to the test set (unseen data) the predict function can be used:

A picture containing text

Description automatically generated

* 1. We can then generate a confusion matrix to compare the predicted labels against the actual labels. This also generates model metrics, the most important of which are accuracy, kappa, sensitivity, and specificity.

Text

Description automatically generated

At first sight our model looks good – we have achieved 92% accuracy- great! However, a closer look at the confusion matrix shows that our models predicted NO instances of the positive class. 3003 instances of the overseas\_airport Palma (with a value of 1) were incorrectly labelled as 0.

A screenshot of a computer

Description automatically generated with low confidence

|  |  |  |
| --- | --- | --- |
|  | Actual 1 | Actual 0 |
| Predicted 1 | True Positive | False Positive |
| Predicted 0 | False Negative | True Negative |

# Addressing Class Imbalance

This is a typical class imbalance problem, whereby the negative class (not PMI, 0) is much larger than then the positive class (PMI, 1). In this case the positive class, the one we are interested in only contributes 8% of the data. Therefore, the algorithm labels all unseen records as 0 to minimise the error and achieves 92% accuracy. As it turns out accuracy is a terrible metric to judge our model.

* 1. To address the class imbalance problem we can revisit our training set and undertake sampling to reduce the size between the two classes. There are a few alternative ways of doing this, but today we will simply under sample the larger class. NOTE: we will not resample the testing dataset because we want to evaluate how effectively the model will predict unseen data in the real world. So, the testing remains imbalanced.

Graphical user interface, text

Description automatically generated

* 1. Once resampled, the algorithm can be retrained, the unseen data predicted and the confusion matrix generated:

Text

Description automatically generated

The new confusion matrix looks better (1784 /3003 PMI instances have now been correctly labelled), but it could be improved further. One way of doing so would be to run a more complex training algorithm, such as repeated CV. Unfortunately, we don’t have time to do this today but the code is included at the end of the script if you want to run this in your own time. We could also add further variables or examine more advanced sampling techniques such as the Synthetic Minority Oversampling Technique (SMOTE) (https://www.rdocumentation.org/packages/DMwR/versions/0.4.1/topics/SMOTE). SMOTE involves adding synthetic data points to increase the size of the minority class, however as we are already working with a synthetic dataset this wouldn’t be advised in this case.

A screenshot of a computer

Description automatically generated with low confidence

# Calculating Variable Importance

* 1. In the final part of the practical we will examine highly predictive variables in the model. Caret has an inbuilt function called varimp which can be used to print the variables. Here the train object is passed in.



* 1. We can also pass the output of varimp to a plotting function such as dotplot()



Chart

Description automatically generated

* 1. The most predictive variables for people visiting Palma Mallorca are now shown with age 65+ the most predictive variable. NOTE: the variable importance scores do not tell us anything about the direction of association.
  2. The last part of the script uses ggplot to create a more aesthetic version of the graph:

Graphical user interface, text, application, email

Description automatically generated

GGplot output:

Chart

Description automatically generated

# Optional Task

Build a model to predict whether a person is likely to travel to Ibiza based on their socio-demographic characteristics. The code will be very similar but instead of selecting the overseas\_airport\_PMI variable on line 73, you should select the overseas\_airport\_IBZ column.