DMML FINAL PROJECT

Royston Pinto

MSc in Computing(Majors Data Analytics), DCU

royston.pinto2@mail.dcu.ie

Datasets:

1. Bank Marketing Dataset: Bank Marketing Dataset (kaggle.com)
2. Lead Scoring Dataset: Lead Scoring Dataset (kaggle.com)
3. Adult Census Income : Adult Census Income (kaggle.com)

Machine LearningAlgorithms Used:

The above datasets contain a mix of numerical features and categorical features, which shows the existence of potential non-linear relationships between the features. Hence to handle these conditions and to get the complex patterns KNN classification is suitable. Additionally, the datasets are robust, and there are multiple features. So, to handle this robustness and high dimensionality random forest classification is a great option.

Problem Description:

Bank Marketing Dataset:
Here we are finding out whether a client has subscribed to a term deposit or not. This dataset contains all the information collected during a marketing campaign run by a banking institution. It mainly contains the demographic, economic and behavioral attributes like age, job, marital status etc. The insights from this data will help banks to improve their marketing strategies and also it will help to understand consumer behavior.

Lead Scoring Dataset
Here we must identify the leads who have more chances of being converted into a paying customer. Here the dataset mainly contains information about how leads were generated and interaction of leads with the educational institution/salesmen. Analyzing this dataset and finding patterns can help the educational institution to prioritize leads based on the important factors and optimize their marketing strategies.

Adult Census Income:
In the second dataset, we are going to classify the individuals based on their income level. The dataset contains demographic and socioeconomic information about the individuals from United States. The insights from this will help various stakeholders like policymakers, economists, employers, and individuals of United States to make good, informed decisions about economic policies, hiring practices, personal finance etc.

(Answers to the Questions are written after 4 implementations of each dataset. They are in bold)

Characteristics of Datasets:

1. **Bank Marketing Dataset**

Characteristics	Description
i. Variables	Independent Variables: 11 age, job, marital, education, default, balance, housing, loan, contact, day, month, duration, campaign, pdays, previous, poutcome Dependent Variable: 1 deposit
ii. Records	Total Records Before Cleaning: 11162
iii. Data Types	Binary: default, housing, loan, deposit Nominal: job, marital, education, contact, month, poutcome Numerical: age, balance, day, duration, campaign, pdays, previous

iv. Summary Statistics	Mean age: 41 years Average contacts during campaign: 2.5 times Average interaction duration: 372 seconds Extended summary including mean, quartiles for each numeric variables provided after data loading in the code

v. Data Cleaning	Irrelevant Variables Removed: pdays Duplications: There were no duplicate records. Missing Values: There were no missing values. Removed records with negative balance Outliers: 1216 Outliers were there. Treated using IQR. One-Hot Encoding and Label encoding was done for categorical variables.
Data After Cleaning	Independent Variables: 52 as one-hot encoding was done. Dependent Variable: 1 deposit Records Count: 10474
vi. Data Normalization	Used MinMaxScaler() in python and scale() in R to normalize the data
vii. Data Balancing and Splitting	Initial Split: Data split into training (70%) and testing (30%) sets. Balancing: The data was almost balanced i.e. values in column 'deposit' has value as 'no'= 52.6% and 'yes'=47.38% Training Records= 7331 Testing Records = 3143

2. **Adult Census Income:**

Characteristics	**Description**
i. Variables	Independent Variables: 14 age, workclass, fnlwgt, education, education.num, marital.status, occupation, relationship, race, sex, capital.gain, capital.loss, hours.per.week, native.country Dependent Variable: 1 income
ii. Records	Total Records Before Cleaning: 32561
iii. Data Types	Categorical: workclass, education, marital.status, occupation, relationship, race, native.country, income Numerical: age, fnlwgt, education.num, capital.gain, capital.loss, hours.per.week
iv. Summary Statistics	Mean age: 38.58 years Mean working hours per week: 40.44 Extended summary including mean, quartiles for each numeric variables provided after data loading in the code
v. Data Cleaning	Irrelevant Variables Removed: 'capital.gain', 'capital.loss' Duplications: There were 24 duplicate records. Missing Values: There were 4262 missing values. Imputed with mode. Outliers: 757 Outliers were there. Treated using IQR. One-Hot Encoding and Label encoding was done for categorical variables.

Data After Cleaning	Independent Variables: 12 age, workclass, fnlwgt, education, education.num, marital.status, occupation, relationship, race, sex, hours.per.week, native.country Dependent Variable: 1 income Records Count: 32537
vi. Data Normalization	Used MinMaxScaler() in python and scale() in R to normalize the data
vii. Data Balancing and Splitting	Initial Split: Data split into training (70%) and testing (30%) sets. Balancing: The data was imbalanced. Used RandomOversampler to balance the data. Training data= 34484 Testing data = 9762

3. **Lead Score Dataset**

Characteristics	**Description**
i. Variables	Independent Variables: 31 Lead Origin, Lead Source, Do Not Email, Do Not Call, TotalVisits, Total Time Spent on Website, Page Views Per Visit, Last Activity, Country, Specialization, How did you hear about X Education, What is your current occupation, Search, Magazine, Newspaper Article, X Education Forums, Newspaper, Digital Advertisement, Through Recommendations, Receive More Updates About Our Courses, Tags, Lead Quality, Update me on Supply Chain Content, Get updates on DM Content, Lead Profile, City, Asymmetrique Activity Index, Asymmetrique Profile Index, Asymmetrique Activity Score,

	Asymmetrique Profile Score, I agree to pay the amount through cheque, A free copy of Mastering The Interview. Dependent Variable: 1 Converted
ii. Records	Total Records Before Cleaning: 9240
iii. Data Types	Binary: Do Not Email, Do Not Call, Search, Magazine, Newspaper Article, X Education Forums, Newspaper, Digital Advertisement, Through Recommendations, Receive More Updates About Our Courses, Update me on Supply Chain Content, Get updates on DM Content, I agree to pay the amount through cheque, A free copy of Mastering The Interview Categorical : Lead Origin, Lead Source, Last Activity, Country, Specialization, How did you hear about X Ordinal : Asymmetrique Activity Index, Asymmetrique Profile Index Education, What is your current occupation, City, Tags, Lead Profile, Last Notable Activity Numerical: TotalVisits, Total Time Spent on Website, Page Views Per Visit, Asymmetrique Activity Score, Asymmetrique Profile Score Text : Prospect ID, Lead Number
iv. Summary Statistics	Percentage of converted leads: 38.54% Extended summary including mean, quartiles for each numeric variables provided after data loading in the code
v. Data Cleaning	Irrelevant Variables Removed: 'How did you hear about X Education', 'Lead Quality', 'Lead Profile', 'Asymmetrique Activity Index', 'Asymmetrique Profile Index', 'Asymmetrique Activity Score', 'Asymmetrique Profile Score', ‘Newspaper Article’, ’Do Not Email’,

	’Do Not Call’, ‘What matters most to you in choosing a course’, ’Search’, ’Magazine’, ’X Education Forums’, ’Newspaper’, ’Digital Advertisement’, ’Through Recommendations’, ’Receive More Updates About Our Courses’, ’Update me on Supply Chain Content’, ’Get updates on DM Content’, ’I agree to pay the amount through cheque’, ‘Tags’, ‘Prospect ID’, ’Lead Number’, ’City’ Duplications: There were no duplicate records. Missing Values: There were 37821 missing values. Imputed with mode. Rows Dropped : 166 Outliers: 170 Outliers were there. Treated using IQR. One-Hot Encoding and Label encoding was done for categorical variables.
Data After Cleaning	Independent Variables: 52 due to one-hot encoding Dependent Variable: 1 Converted Records Count: 9074
vi. Data Normalization	Used MinMaxScaler() in python and scale() in R to normalize the data
vii. Data Balancing and Splitting	Data split into training (70%) and testing (30%) sets. Balancing: The data was imbalanced. Used RandomOversampler to balance the data. Training data= 7954 Testing data = 2723

Model Building:

1. Bank Marketing Dataset:

Here before the model building the cleaned data is loaded and the data is normalized. For python implementations we are using MinMaxScaler() and in R we are using preprocess() function. Next the data is split into training and testing with ratio of 0.7 and 0.3. In this dataset, the data is almost balanced. Hence no need of using any balancing technique.

Implementing K-Nearest Neighbors:

a) Implementation of K-Nearest Neighbors Classification with KNeighborsClassifier using Python's Scikit-learn library :

Here we have used GridSearch cross-validation technique to find the best parameters for the model. Here cross-validation was conducted for k value. We have kept the other parameters like 'metric' as 'euclidean' and 'weight' as 'uniform'. Through the cross-validation the best value for k is 9. While building knn model in R also we have kept the same k value and by default R has 'metric' as 'euclidean' and 'weight' as 'uniform'.

By setting these parameters the model was built for our data.

Below are the evaluation metrics:

```
Testing Accuracy of kNN with best hyperparameters: 0.7018771874005727
Confusion Matrix:
[[1214  395]
 [ 542  992]]
Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.75      0.72      1609
           1       0.72      0.65      0.68      1534

    accuracy                           0.70      3143
   macro avg       0.70      0.70      0.70      3143
weighted avg       0.70      0.70      0.70      3143

ROC AUC Score: 0.7720625831069207
```

In this implementation we were able to get a model accuracy of 0.70 and f1-score of 0.7.

b) Implementation of K-Nearest Neighbors Classification using R's caret library.

Here while building knn model in R we have kept the same k value i.e. 9 and by default R has 'metric' as 'euclidean' and 'weight' as 'uniform'. So there will be consistency between these two implementations.

Evaluation metrics here:

```
[1] "Evaluation Metrics:"
> print(paste("Testing Accuracy:", accuracy))
[1] "Testing Accuracy: 0.699013681196309"
> print(paste("Precision:", precision))
[1] "Precision: 0.644299674267101"
> print(paste("Recall:", recall))
[1] "Recall: 0.712023038156947"
> print(paste("F1-Score:", f1_score))
[1] "F1-Score: 0.676470588235294"
```

Here we were able to achieve a model accuracy of 0.699 and F1-score of 0.67.

c) Implementation of Random Forest Classification with RandomForestClassifier using Python's Scikit-learn library :

Here we have used GridSearch cross-validation(5 fold cross-validation) technique to find the best hyperparameters for the model.
Best hyperparameters: 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200

Evaluation metrics:

```
Testing Accuracy of Random Forest with best hyperparameters: 0.844097995545657
Confusion Matrix:
[[1323  286]
 [ 204 1330]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.82      0.84      1609
           1       0.82      0.87      0.84      1534

    accuracy                           0.84      3143
   macro avg       0.84      0.84      0.84      3143
weighted avg       0.85      0.84      0.84      3143

ROC AUC Score: 0.7720625831069207
```

Here we can see that using Random Forest Classifier, we were able to get accuracy of 0.84 and f1-score of 0.84.

d) Implementation of Random Forest Classification using R's caret library:
Here we have kept same parameters as the Python's implementation with slight syntax differences.

ntree = 200 (Number of trees)
max_depth = 20(Maximum depth of the trees)
min_node_size = 2 (Minimum number of samples required to be at a leaf node)

Evaluation metrics:

```
> print("Evaluation Metrics:")
[1] "Evaluation Metrics:"
> print(paste("Testing Accuracy:", accuracy))
[1] "Testing Accuracy: 0.840598154629335"
> print(paste("Precision:", precision))
[1] "Precision: 0.865798045602606"
> print(paste("Recall:", recall))
[1] "Recall: 0.818349753694581"
> print(paste("F1-Score:", f1_score))
[1] "F1-Score: 0.841405508072175"
```

Here we can see the F1-Score is 0.84 and Accuracy is 0.84.

From the above we say that KNN models implemented in Python's Scikit-learn and R's caret library both perform equally with accuracy of 0.7. Similarly random forest models implemented in Scikit-Learn and Caret library also perform equally.
Out of these 4 we may select Random Forest model implemented using Scikit-Learn library. The reason for this may be that random forest uses ensemble approach which reduces the overfitting and improve the accuracy. Also, random forest algorithm is less sensitive to outliers which might also be the reason.

2. Adult Census Income

Here before the model building the cleaned data is loaded and the data is normalized. For python implementations we are using MinMaxScaler() and in R we are using preprocess() function. Next the data is split into training and testing with ratio of 0.7 and 0.3. For data balancing we have used RandomOverSampler from imbalance-learn library.

Implementing K-Nearest Neighbors:

a) Implementation of K-Nearest Neighbors Classification with KNeighborsClassifier using Python's Scikit-learn library :

Here we have used GridSearch cross-validation technique to find the best parameters for the model. Here cross-validation was conducted for k value. We have kept the other parameters like 'metric' as 'euclidean' and 'weight' as 'uniform'. Through the cross-validation the best value for k is 1. While building knn model in R also we have kept the same k value and by default R has 'metric' as 'euclidean' and 'weight' as 'uniform'.

By setting these parameters the model was built for our data.

Below are the evaluation metrics:

```
Testing Accuracy of kNN with best hyperparameters: 0.7846752714607662
Confusion Matrix:
[[6373 1083]
 [1019 1287]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.85      0.86      7456
           1       0.54      0.56      0.55      2306

    accuracy                           0.78      9762
   macro avg       0.70      0.71      0.70      9762
weighted avg       0.79      0.78      0.79      9762
```

In this implementation we were able to get a model accuracy of 0.78 and f1-score of 0.79.

b) Implementation of K-Nearest Neighbors Classification using R's caret library.

Here while building knn model in R we have kept the same k value i.e. 1 and by default R has 'metric' as 'euclidean' and 'weight' as 'uniform'. So there will be consistency between these two implementations.

Evaluation metrics here:

```
[1] "Evaluation Metrics:"
> print(paste("Testing Accuracy:", accuracy))
[1] "Testing Accuracy: 0.776172915386191"
> print(paste("Precision:", precision))
[1] "Precision: 0.533988884138521"
> print(paste("Recall:", recall))
[1] "Recall: 0.532849829351536"
> print(paste("F1-Score:", f1_score))
[1] "F1-Score: 0.533418748665386"
```

Here we were able to achieve a model accuracy of 0.776 and F1-score of 0.53.

c) Implementation of Random Forest Classification with RandomForestClassifier using Python's Scikit-learn library :

Here we have used GridSearch cross-validation(5 fold cross-validation) technique to find the best hyperparameters for the model.
Best hyperparameters: 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200

Evaluation metrics:

```
Testing Accuracy of Random Forest with best hyperparameters: 0.8098750256095062
Confusion Matrix:
[[6257 1199]
 [ 657 1649]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.84      0.87      7456
           1       0.58      0.72      0.64      2306

    accuracy                           0.81      9762
   macro avg       0.74      0.78      0.76      9762
weighted avg       0.83      0.81      0.82      9762
```

Here we can see that using Random Forest Classifier, we were able to get accuracy of 0.81 and f1-score of 0.82.

d) Implementation of Random Forest Classification using R's caret library:
Here we have kept same parameters as the Python's implementation with slight syntax differences.

ntree = 200 (Number of trees)
max_depth = 20(Maximum depth of the trees)
min_node_size = 2 (Minimum number of samples required to be at a leaf node)

Evaluation metrics:

```
> print("Evaluation Metrics:")
[1] "Evaluation Metrics:"
> print(paste("Testing Accuracy:", accuracy))
[1] "Testing Accuracy: 0.807928703134604"
> print(paste("Precision:", precision))
[1] "Precision: 0.769987174005985"
> print(paste("Recall:", recall))
[1] "Recall: 0.573932441045252"
> print(paste("F1-Score:", f1_score))
[1] "F1-Score: 0.657659302537886"
```

Here we can see that using Random Forest Classifier, we were able to get accuracy of 0.8 and f1-score of 0.65.

1. The results from above show that there are differences between the implementations.
2. Here its evident that KNN model in python scikit learn outperforms R's implementation. Its superior in both accuracy and f1-score. Its same in the case of Random forest algorithm. Its Python implementation is more accurate than R's implementation. Here R library fails in determining the true positive values. Out of these 4 Random Forest Scikit-Learn Implementation is best for this dataset.

3. **Lead Scoring Dataset**
Here before the model building the cleaned data is loaded and the data is normalized. For python implementations we are using MinMaxScaler() and in R we are using preprocess() function. Next the data is split into training and testing with ratio of 0.7 and 0.3. For data balancing we have used RandomOverSampler from imbalance-learn library.

a) Implementation of K-Nearest Neighbors Classification with KNeighborsClassifier using Python's Scikit-learn library :

Here we have used GridSearch cross-validation technique to find the best parameters for the model. Here cross-validation was conducted for k value. We have kept the other parameters like ‘metric’ as ‘euclidean’ and ‘weight’ as ‘uniform’. Through the cross-validation the best value for k is 1. While building knn model in R also we have kept the same k value and by default R has ‘metric’ as ‘euclidean’ and ‘weight’ as ‘uniform’.

By setting these parameters the model was built for our data.

Below are the evaluation metrics:

In this implementation we were able to get a model accuracy of 0.72 and f1-score of 0.72.

b) Implementation of K-Nearest Neighbors Classification using R's caret library.

Here while building knn model in R we have kept the same k value i.e. 1 and by default R has ‘metric’ as ‘euclidean’ and ‘weight’ as ‘uniform’. So there will be consistency between these two implementations.
Evaluation metrics here:

```
> print("Evaluation Metrics:")
[1] "Evaluation Metrics:"
> print(paste("Testing Accuracy:", accuracy))
[1] "Testing Accuracy: 0.751377157546823"
> print(paste("Precision:", precision))
[1] "Precision: 0.618525896414343"
> print(paste("Recall:", recall))
[1] "Recall: 0.678688524590164"
> print(paste("F1-Score:", f1_score))
[1] "F1-Score: 0.647212089630016"
```

Here we were able to achieve a model accuracy of 0.75 and F1-score of 0.64.

c) Implementation of Random Forest Classification with RandomForestClassifier using Python's Scikit-learn library :

Here we have used GridSearch cross-validation(5 fold cross-validation) technique to find the best hyperparameters for the model.
Best hyperparameters: 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100

Evaluation metrics:

```
Testing Accuracy of Random Forest with best hyperparameters: 0.8053617333822989
Confusion Matrix:
[[1391  271]
 [ 259  802]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.84      0.84      0.84      1662
         1.0       0.75      0.76      0.75      1061

    accuracy                           0.81      2723
   macro avg       0.80      0.80      0.80      2723
weighted avg       0.81      0.81      0.81      2723
```

Here we can see that using Random Forest Classifier, we were able to get accuracy of 0.81 and f1-score of 0.81.

d) Implementation of Random Forest Classification using R's caret library:
Here we have kept same parameters as the Python's implementation with slight syntax differences.

ntree = 100 (Number of trees)
max_depth = 20(Maximum depth of the trees)
min_node_size = 2 (Minimum number of samples required to be at a leaf node)

Evaluation metrics:

```
[1] "Evaluation Metrics:"
> print(paste("Testing Accuracy:", accuracy))
[1] "Testing Accuracy: 0.827029012118986"
> print(paste("Precision:", precision))
[1] "Precision: 0.780876494023904"
> print(paste("Recall:", recall))
[1] "Recall: 0.757487922705314"
> print(paste("F1-Score:", f1_score))
[1] "F1-Score: 0.769004413928396"
```

Here the accuracy is 0.82 and f1-score is 0.76.

1.In terms of accuracy both implemenations of knn are equally good. But in terms of f1 score Knn of scikit learn outperforms knn model of R. This is same in the case of Random Forest.
2.Out of these Random Forest Scikit-Learn implementation is best with both accuracy and f1-score.

Adult Census Income
Below are the results:
Result of Knn model using python scikit learn:
Accuracy=0.78 F1-score=0.79

Result of Knn model using R caret:
Accuracy=0.7 7 F1-score=0.53

Result of Random Forest model using python scikit learn:
Accuracy=0.81 F1-score=0.82

Result of Random Forest R caret:
Accuracy=0.84 F1-score=0.65