WEEK 13: Modelling

Name: Sainad Reddy Naini (Individual)
Email: nainisainad@gmail.com
Country: India
College: University of Bath (Graduate -UK)
Specialization: Data Science

1: PROBLEM DESCRIPTION

ABC Bank is preparing to launch a new term deposit product and aims to maximize the effectiveness of its marketing efforts. To achieve this, they want to develop a machine learning (ML) model that can predict whether a customer will subscribe to the term deposit based on historical data from previous marketing campaigns. This predictive model will help the bank identify potential customers who are more likely to purchase the product, allowing them to focus their marketing efforts more effectively and efficiently.

Note: Refer to the previous week's materials for an idea of the data problems and how we addressed them, as well as exploratory data analysis (EDA). In this section, we will see how to build the model, fine-tune it, and create a final model for predictions, along with suggestions for improvement.

2: GITHUB REPO LINK

https://github.com/sainadreddy/Data_Glacier_Intership_2024

3: MODELLING

3.1: Converting data

Convert all the data into numerical format because machine learning algorithms can only work with numerical data. This step ensures that the data is ready to be processed by the models. Once the data is converted, we will clean and prepare it for training the models.

First, we will remove all the extra columns added during EDA for visualizations, specifically **y_numeric** and **Age Group.**

Second, we will use two methods to convert categorical columns into numeric format:

- **One-hot encoding**: We will use this for all categorical columns except for the target variable (y) because it helps avoid confusion for the ML model, although it adds more columns.
- **Label encoding:** The above method creates separate columns for each category, which we don't need for the target variable because we only require one value for prediction. Therefore, we will use this for the target variable.

Note: We perform this using the duration feature and without the duration feature to make a comparison at the end, not to create a predictive model. As stated before, we cannot use the duration feature for prediction, as it cannot be known before making the call.

3.2: Splitting the data

We will divide the dataset into two parts: one for training the models and another for testing how well they work. The training set helps the models learn, while the testing set checks their performance on new data. This way, we can evaluate whether the models are effective at making predictions. We will use an **80-20%** split for training and testing, as this is an ideal ratio.

```
# Define features and target variable
X = df_clean.drop(columns=['y'])
y = df_clean['y']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3.3: Handling imbalance

To deal with the imbalance in the data, I will use **SMOTE** (Synthetic Minority Over-sampling Technique). This method creates new examples for the less common category instead of removing some from the more common category. I chose SMOTE because it helps keep more information in the dataset, which can lead to better predictions from the model.

```
# Apply SMOTE for oversampling
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

Note: SMOTE is applied only to the training data, not the testing data, because the test set should remain untouched and should only contain real, original data. This allows for an honest evaluation.

3.4: Model Exploration

Since our problem involves predicting a binary outcome (yes or no) and we have imbalanced data, I have chosen the following models: -

- **Logistic Regression (linear)**: A simple model that is easy to understand. It may not capture complex patterns well, so we will use it as a starting point.
- **Decision Tree (non-linear)**: Handles both numeric and categorical values and helps identify important features, but it may overfit the data.
- **Random Forest (Ensemble):** Solves the overfitting issue in decision trees and manages imbalanced data by adjusting class weights.
- **XGBoost (Boosting)**: Similar to Random Forest but generally more accurate, especially in dealing with complex relationships.

3.4.1 Metrics used to measure performance

- **Accuracy**: The percentage of all predictions (both correct and incorrect) that the model got right.
- **Precision**: The percentage of correct positive predictions made by the model out of all positive predictions it made.
- **Recall**: The percentage of actual positives that were correctly identified by the model.
- **F1-Score**: A balance between precision and recall, showing how well the model performs on positive cases.
- **ROC AUC**: A measure of how well the model distinguishes between positive and negative cases, with a score closer to 1 indicating better performance.
- **Confusion Matrix**: A table that shows the counts of correct and incorrect predictions broken down by actual and predicted classes.

3.4.2 Observations without duration feature

- **Accuracy:** Both the Random Forest model and XGBoost performed well, correctly predicting around 88.5% of the cases.
- **Precision:** XGBoost was the best at correctly identifying positive cases, with about 48.5% of its positive predictions being accurate.
- **Recall:** The Random Forest model was better at finding actual positive cases, catching about 43.6% of them.
- **F1-Score:** Random Forest also balanced precision and recall the best, scoring 45.7% overall.
- **ROC AUC:** Random Forest showed the best ability to distinguish between the two classes, with a score of 0.6888.

Conclusion: The Random Forest model seems to work the best among all the options we tried, providing a good balance of performance. The next step will be to fine-tune this model to make it even better.

3.4.3 Observations with duration feature

- **Accuracy:** Both the Random Forest model and XGBoost performed well, correctly predicting around 91% of the cases.
- **Precision:** XGBoost was the best at correctly identifying positive cases, with about 60.44% of its positive predictions being accurate.
- **Recall:** The XGBoost model was better at finding actual positive cases, catching about 61.24% of them.

- **F1-Score:** XGBoost also balanced precision and recall the best, scoring 60.84% overall.
- **ROC AUC:** Random Forest and XGBoost showed the best ability to distinguish between the two classes, with a score of 0.7812 and 0.7795 respectively.

Conclusion: The XGBoost model seems to work the best among all the options we tried, providing a good balance of performance.

3.4.4 Comparing model performance with and without duration feature

Overall Insights

- The introduction of the duration feature has positively impacted the performance of all models, particularly Random Forest and XGBoost, leading to higher accuracy, precision, recall, F1-Score, and ROC AUC.
- **XGBoost** consistently shows improvements and currently outperforms the other models across all key metrics, indicating it is the most effective model when including the duration feature.
- The significant gains in recall and precision across the board suggest that the models are becoming more reliable in their predictions, especially in identifying positive cases.

Conclusion

The inclusion of the duration feature has enhanced the models' capabilities, particularly benefiting XGBoost, which now stands out as the top performer. Further tuning and optimization may yield even better results. However, since we are not using the duration feature in our predictions, we will use Random Forest and fine-tune it.

3.5: Model Selection with parameter tuning.

Since we have chosen our final model, which is the **Random Forest classifier**, let's proceed with hyperparameter tuning. We have two methods to do this:

- **Grid Search CV**: A slow and expensive method.
- **Random Search CV**: A faster and more efficient method.

I have used Randomized Search CV, but you could first try Randomized Search CV to get a range of parameters and then use Grid Search CV to explore every combination to find the best parameters.

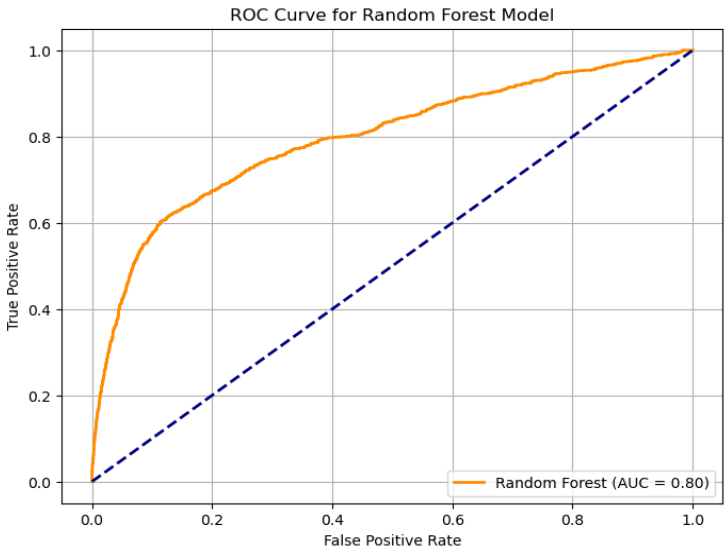
After trying multiple combinations and using trial and error, I have selected the following parameters:

```
param_grid = {
    'n_estimators': [172,173,174,175,176,177,178,179,180],
    'max_depth': [24,25,26,27,28,29,30],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [2,3,4,5],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False],
    'class_weight': [None, 'balanced']
}
```

After fine-tuning, I obtained the following model:

```
RandomForestClassifier(bootstrap=False, class_weight='balanced', max_depth=27,
                       max_features='log2', min_samples_leaf=2,
                       n_estimators=178, random_state=42)
```

This model achieved an accuracy of 88.65%, which is a slight increase compared to the Random Forest model without tuning. Additionally, all the other metrics, such as F1-Score, precision, and Recall, improved as well. This is important since we are dealing with imbalanced data, and accuracy is not the only metric we need to monitor to ensure a good model.



To test the model, I used the AUC ROC Curve and obtained a value of around 0.8, which suggests that the model does not make random guesses and makes significantly better decisions, correctly ranking or classifying a large portion of instances. However, it may not be perfect and could still miss some positive instances or incorrectly identify some negatives as positives. Overall, the model is performing better, but additional fine-tuning is required to achieve optimal performance.

4: FINAL RECOMMENDATION

Further fine-tuning of the model parameters is recommended. Unfortunately, due to limited computational power, I was unable to conduct a more extensive tuning process. However, the current model still performs well and can be used to make predictions for future customers.

As highlighted during the EDA (Exploratory Data Analysis), it is advisable to focus on retirees and single individuals for term deposit campaigns, prioritize cell phone outreach, and target peak periods in December, March, October, and September. Additionally, higher engagement is observed on Thursdays and Tuesdays, so marketing efforts should be concentrated on these days.