Development of Regular and Recursive Functions in Predictive Analytics on nursing home investment

Task 3 - by TEAM 6

Joshua Cabal
Namrata Patil
Venkat-Amit Kommineni
Denise Becerra
Krish Viswanadhan-Nair

INTRODUCTION - PROJECT OVERVIEW

Nursing Home

Due to the significant number of baby boomers exploring nursing home options, our company has been tasked with determining if it is advisable for our client to invest in nursing homes. As a member of the data analysis team, we are required to perform an analysis to address this question using iterative and recursive functions.

Our team used predictive algorithms to assess the viability of investments and offer financial advice to clients who were interested in investing in nursing homes in the United States. We employed feature engineering to generate significant financial ratios and we used a mix of these ratios to generate the target variable called the "Investment Choice" , and we processed the data to resolve missing values, outliers, and inconsistencies using cost report data from 2015–2021, collected from CMS. We used PCA and scaling techniques to control the dimensionality of the data and normalize the inputs. Three machine learning models (KNN, Random Forest, and Logistic Regression) were chosen, trained on 75% of the data, and assessed with 75% of the data going toward training, 10% going toward validation, and 15% going toward testing. Accuracy, precision, recall, F1 score, confusion matrix, and ROC AUC were used to evaluate the performance of the model, and cross-validation was used to improve the model and reduce overfitting. The Random Forest model was refined and implemented in a production setting because it showed the best accuracy and AUC ROC. Using predictive analysis we can help our clients decide if they want to invest in nursing homes.We used Regular function and Recursive function in all of the following steps:

1. Data Preprocessing
2. Feature selection
3. Model Building
4. Model evaluation

METHODOLOGY REGULAR VS RECURSIVE FUNCTION

We used Regular functions for most of the predictive analytics steps because of their simple logic and effectiveness in the majority of situations. Recursive functions can be handy when there is a natural way to break an issue down into smaller, related subproblems.

1. def normalize_column_names(df): was used to normalize column names by removing data inconsistency, where space was replaced by underscore between variable names.

def normalize_column_names_recursive(df, index=0): this function recursively finds the variables with space between the names and then replaces them with underscore.

Data Cleaning - Regular Function ¶

```
def normalize_column_names(df):
    df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
    return df
```

Data Cleaning - Recursive Function

```
import pandas as pd

def normalize_column_names_recursive(df, index=0):
    if index == len(df.columns):  # Base case: All columns are processed
        return df

    # Normalize the current column name
    old_name = df.columns[index]
    new_name = old_name.strip().lower().replace(' ', '_')
    df = df.rename(columns={old_name: new_name})

    # Recursive call for the next column
    return normalize_column_names_recursive(df, index + 1)
```

2.def calculate_investment_choice(df): This function iteratively calculates our target variable Investment choice using the financial ratios.

def calculate_investment_choice_recursive(): Recursively calculates the target variable by setting the financial ratios based on threshold.

Feature Engineering- Regular Function

```
def calculate_investment_choice(df):

    condition1 = ((df['ROI'] >ROI_Threshold) | (df['Operating_Margin'] > OpMargin_Threshold))
    condition2 = ((df['Debt_to_equity_ratio'] < DTE_Threshold) | (df['Current_Ratio'] >CR_Threshold))
    condition3 = ((df['Days_in_Accounts_Receivable'] < DAR_Threshold) | (df['Bed_Occupancy_Rate'] > BOR_Threshold))

    # Combine all conditions using logical AND
    final_condition = condition1 & condition2 & condition3

    # Apply the final condition to create the 'Investment_Choice' column
    df['Investment_Choice'] = np.where(final_condition, 1, 0)
    return df
```

Feature Engineering- Recursive function

```
import pandas as pd
import numpy as np

def calculate_investment_choice_recursive(df, index=0, ROI_Threshold=0, OpMargin_Threshold=0, DTE_Threshold=0, CR_Threshold=0, I
    # Base case: if index is equal to the length of the DataFrame, stop recursion
    if index == len(df):
        return df
    # Conditions for the current row
    condition1 = (df.loc[index, 'ROI'] > ROI_Threshold) or (df.loc[index, 'Operating_Margin'] > OpMargin_Threshold)
    condition2 = (df.loc[index, 'Debt_to_equity_ratio'] < DTE_Threshold) or (df.loc[index, 'Current_Ratio'] > CR_Threshold)
    condition3 = (df.loc[index, 'Days_in_Accounts_Receivable'] < DAR_Threshold) or (df.loc[index, 'Bed_Occupancy_Rate'] > BOR_Th
    # Combine conditions
    final_condition = condition1 and condition2 and condition3
    # Assign the result to the 'Investment_Choice' column
    df.loc[index, 'Investment_Choice'] = 1 if final_condition else 0
    # Recursive call for the next row
    return calculate_investment_choice_recursive(df, index + 1, ROI_Threshold, OpMargin_Threshold, DTE_Threshold, CR_Threshold,
# Example usage:
data = pd.DataFrame({
    'ROI': [10, 20, 5],
    'Operating_Margin': [15, 25, 5],
    'Debt_to_equity_ratio': [0.5, 0.3, 0.7],
    'Current_Ratio': [1.5, 2, 1],
    'Days_in_Accounts_Receivable': [30, 45, 25],
    'Bed_Occupancy_Rate': [80, 90, 75]
})
# Assume some thresholds
data = calculate_investment_choice_recursive(data, ROI_Threshold=10, OpMargin_Threshold=10, DTE_Threshold=0.6, CR_Threshold=1.2
print(data)
```

3. def evaluate_logistic_regression(X_train_pca, y_train, X_test_pca, y_test, X_val_pca, y_val): This function creates a Classification Model to ascertain investment eligibility based on predetermined financial health standards.

def recursive_evaluate_logistic_regression(log_reg, datasets, index=0): This function recursively creates the logistic regression model based on the train set, test set and the validation set.

Logistic Regression - Regular Function

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

def evaluate_logistic_regression(X_train_pca, y_train, X_test_pca, y_test, X_val_pca, y_val):
    # Initialize the Logistic Regression model
    log_reg = LogisticRegression(max_iter=1000, solver='sag', random_state=42)

    # Train the model
    log_reg.fit(X_train_pca, y_train)

    # Evaluation function to predict and print accuracy
    def predict_and_print_accuracy(X, y, set_name):
        y_pred = log_reg.predict(X)
        accuracy = accuracy_score(y, y_pred)
        print(f"{set_name} set accuracy:", accuracy)

    # Evaluate on training, test, and validation sets
    predict_and_print_accuracy(X_train_pca, y_train, "Training")
    predict_and_print_accuracy(X_test_pca, y_test, "Test")
    predict_and_print_accuracy(X_val_pca, y_val, "Validation")
```

Logistic Regression - Recursive Function

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

def recursive_evaluate_logistic_regression(log_reg, datasets, index=0):
    if index == len(datasets):
        return  # Base case: no more datasets to process

    X, y, set_name = datasets[index]
    y_pred = log_reg.predict(X)
    accuracy = accuracy_score(y, y_pred)
    print(f"{set_name} set accuracy:", accuracy)

    # Recursive call for the next dataset
    recursive_evaluate_logistic_regression(log_reg, datasets, index + 1)

def main_logistic_regression(X_train_pca, y_train, X_test_pca, y_test, X_val_pca, y_val):
    log_reg = LogisticRegression(max_iter=1000, solver='sag', random_state=42)
    log_reg.fit(X_train_pca, y_train)  # Train the model once

    datasets = [
        (X_train_pca, y_train, "Training"),
        (X_test_pca, y_test, "Test"),
        (X_val_pca, y_val, "Validation")
    ]

    recursive_evaluate_logistic_regression(log_reg, datasets)
```

We have similar functions for KNN and Random Forest Model. All the codes are available in the git repository. (GitHub)

def knn_regular_function(X_train_scaled, y_train, X_test_scaled, y_test, X_val_scaled, y_val, n_neighbors_list):

def knn_regular_function(X_train_scaled, y_train, X_test_scaled, y_test, X_val_scaled, y_val, n_neighbors_list):

def evaluate_model(X_train_scaled, y_train, X_test_scaled, y_test, X_val_scaled, y_val):

def evaluate_model_recursive(rf, datasets, index=0):

In model evaluation we used def plot_roc_curve(y_true, y_prob, model_name): that iteratively calculates the Aucroc values.

We employed three models, KNN, Logistic regression and random forest model to predict the best nursing homes to invest in. Then Model evaluation was done to check the accuracy, precision, recall and we plotted Auc roc curve which gave us the most accurate model to be used.

After the first implementation, we ran the code to make sure it is more readable and efficient. This could entail leveraging built-in functions, reworking to eliminate unnecessary code, or utilizing more effective data structures.We verified that the function functions properly under a range of input conditions, including edge cases. Testing was done by hand. We have provided comments and documentation that describe the parameters, the output, and the operation of the function.

PERFORMANCE COMPARISON

After working on iterative and recursive functions we can see that the recursive functions are time consuming. In our case we can easily work on iterative functions as they are simple, robust and perform slightly better.

CONCLUSION

We have learned a lot from our extensive research on using predictive analytics to finance nursing home investments. Notably, we have identified that using recursive functions for Model building is not particularly fruitful. In our case of data preprocessing , feature engineering, Model employment and model evaluation we can collectively say that iterative functions are simple and work faster than complex recursive functions we tried to implement.

We can conclude that investment of nursing homes opportunities in California,Florida and Ohio, among other states, are the highest. According to our study's predicted models using iterative and recursive functions and historical data, these states provide the highest possible returns. So our client can confidently invest in California for a high return on investment if not any other state.