Predictive Analysis of Bank Marketing Campaigns: A CRISP-DM Approach

September 30, 2023

0.1 Abstract

This paper presents a systematic approach to analyzing bank marketing campaigns using the CRISP-DM methodology. Through data preparation, modeling, and evaluation, we aim to predict the likelihood of a client subscribing to a term deposit. Our findings reveal the importance of data-driven strategies in optimizing marketing efforts and resource allocation.

In today's data-driven era, financial institutions harness the power of data to refine their operations and understand customers' behaviors. One such quest is predicting whether a client will subscribe to a term deposit. In this comprehensive guide, we'll journey through the bank marketing dataset, embracing the structured approach of the CRISP-DM methodology.
Here we are using a Bank Marketing Kaggle data set.
let's embark on the journey of exploring and analyzing the bank marketing dataset using the CRISP-DM methodology. The CRISP-DM (Cross-Industry Standard Process for Data Mining) is a framework consisting of six phases:

1. Business Understanding: Understand the problem and objectives.
2. Data Understanding: Explore and familiarize oneself with the data.
3. Data Preparation: Clean and preprocess the data.
4. Modeling: Build and evaluate models.
5. Evaluation: Assess the performance of the models.
6. Deployment: Implement the model into a production environment.

0.2 Business Understanding: Why Predict?

In the competitive realm of banking, predicting a customer's next move can be a game-changer. By forecasting the outcome of marketing campaigns, banks can strategically allocate resources, tailor their outreach, and ultimately maximize their ROI. Our mission? Predict if a client, when contacted, will subscribe to a term deposit.

0.3 Data Understanding: Peeking into the Data

The dataset serves as a historical record, encompassing attributes like age, job, marital status, and education, intertwined with campaign-specific details. At the heart of the dataset lies the binary target variable: did the client subscribe to a term deposit?
Visual explorations revealed distributions, tendencies, and outliers. For instance, we spotted patterns in job types and their propensity to subscribe.

1 Deeper look at its features:

- age: Age of the client.
- job: Type of job.
- marital: Marital status.
- education: Level of education.
- default: Whether the client has credit in default.
- balance: Average yearly balance in euros.
- housing: Whether the client has a housing loan.
- loan: Whether the client has a personal loan.
- contact: Type of communication used for the last contact.
- day: Last contact day of the month.
- month: Last contact month of the year.
- duration: Duration of the last contact in seconds.
- campaign: Number of contacts performed during this campaign.
- pdays: Number of days since the client was last contacted from a previous campaign.
- previous: Number of contacts performed before this campaign.
- poutcome: Outcome of the previous marketing campaign.
- y: Target variable (whether the client subscribed to a term deposit).

2 Data Understanding: Summary

General Information:

- The dataset has 4,521 entries.
- There are no missing values in any of the columns.

3 Features Overview:

- Numerical Features: ‘age’, ‘balance’, ‘day’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’.
- Categorical Features: ‘job’, ‘marital’, ‘education’, ‘default’, ‘housing’, ‘loan’, ‘contact’, ‘month’, ‘poutcome’, ‘y’.

4 Descriptive Statistics:

- Age: Clients range in age from 19 to 87 years, with a mean age of approximately 41 years.
- Balance: The average yearly balance is around €1422.66. The balance ranges from -€3313 to €71188.
- Duration: The average duration of the last contact is approximately 264 seconds.
- Campaign: On average, a client was contacted approximately 3 times during this campaign.
- Pdays: A significant number of clients have a value of -1 for ‘pdays’, indicating they were not contacted before this campaign.
- Previous: On average, a client was contacted approximately 0.54 times before this campaign.

5 Categorical Data Insights:

- Job: The most common job category is ‘management’.
- Marital: The majority of clients are married.
- Education: Most clients have received a secondary education.
- Housing: A majority of the clients have a housing loan.
- Loan: Most clients do not have a personal loan.
- Contact: The most common communication type is ‘cellular’.
- Month: May is the month with the highest number of last contacts.
- Poutcome: The outcome of the previous campaign is mostly ‘unknown’.
- Y (Target Variable): The majority of clients have not subscribed to a term deposit.

5.1 Data Preparation: Crafting Quality Data

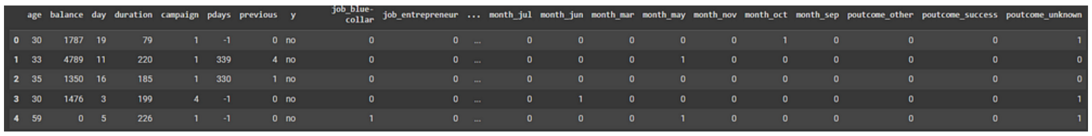
Garbage in, garbage out! Quality data is paramount for insightful models. We undertook several steps:

- Encoding: Life isn't always numerical. Categories like job types were transformed into numbers using one-hot encoding.
- Standardization: To place every feature on an equal footing, we centered them around zero and scaled them to have a unit variance.
- Outlier Management: Extreme values can skew predictions. Techniques like the IQR ensured we kept anomalies in check.

For our dataset:

1. We've already verified that there are no missing values.
2. We'll need to encode the categorical variables.
3. It might be beneficial to standardize the numerical variables.
4. We'll check for outliers and decide how to handle them.
5. We'll explore the potential for feature engineering.

```
# One-hot encoding the categorical variables
bank_data_encoded = pd.get_dummies(bank_data, columns=['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome'], drop_first=True)

# Display the first few rows of the encoded dataset
bank_data_encoded.head()
```



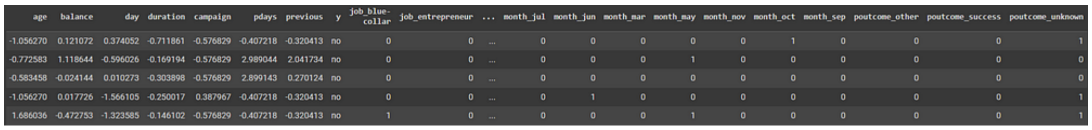
```
   age  balance  day  duration  campaign  pdays  previous   y  job_blue-  job_entrepreneur  ...  month_jul  month_jun  month_mar  month_may  month_nov  month_oct  month_sep  poutcome_other  poutcome_success  poutcome_unknown
                                                                  collar
0   30     1787   19        79         1     -1         0  no          0                 0  ...          0          0          0          0          0          1          0               0                 0                 1
1   33     4789   11       220         1    339         4  no          0                 0  ...          0          0          0          1          0          0          0               0                 0                 0
2   35     1350   16       185         1    330         1  no          0                 0  ...          0          0          0          0          0          0          0               0                 0                 0
3   30     1476    3       199         4     -1         0  no          0                 0  ...          0          1          0          0          0          0          0               0                 0                 1
4   59        0    5       226         1     -1         0  no          1                 0  ...          0          0          0          1          0          0          0               0                 0                 1
```

Here, The categorical variables have been successfully encoded using one-hot encoding. This process has increased the number of columns in our dataset due to the creation of dummy variables.
StandardScaler from sklearn to achieve this.

The numerical features we'll standardize are: 'age', 'balance', 'day', 'duration', 'campaign', 'pdays', and 'previous'.

```
fromsklearn.preprocessing importStandardScaler

# List of numerical features to be standardized
numerical_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']

# Initialize the StandardScaler
```

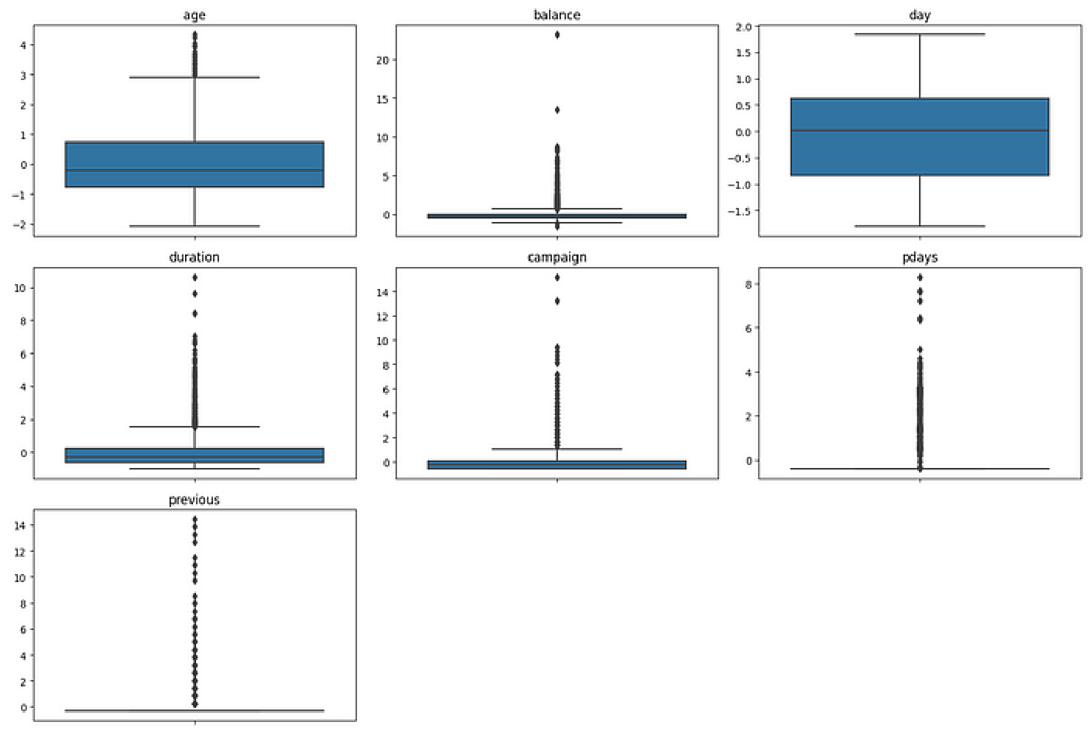
```
scaler = StandardScaler()

# Standardize the numerical features
bank_data_encoded[numerical_features] = scaler.fit_transform(bank_data_encoded[numerical_features])

# Display the first few rows of the standardized dataset
bank_data_encoded.head()
```



age	balance	day	duration	campaign	pdays	previous	y	job_blue-collar	job_entrepreneur	...	month_jul	month_jun	month_mar	month_may	month_nov	month_oct	month_sep	poutcome_other	poutcome_success	poutcome_unknown
-1.056270	0.121072	0.374052	-0.711861	-0.576829	-0.407218	-0.320413	no	0	0	...	0	0	0	0	0	1	0	0	0	1
-0.772583	1.118644	-0.596026	-0.169194	-0.576829	2.989044	2.041734	no	0	0	...	0	0	0	1	0	0	0	0	0	0
-0.583458	-0.024144	0.010273	-0.303898	-0.576829	2.899143	0.270124	no	0	0	...	0	0	0	0	0	0	0	0	0	0
-1.056270	0.017726	-1.566105	-0.250017	0.387967	-0.407218	-0.320413	no	0	0	...	0	1	0	0	0	0	0	0	0	1
1.686036	-0.472753	-1.323585	-0.146102	-0.576829	-0.407218	-0.320413	no	1	0	...	0	0	0	1	0	0	0	0	0	1

The numerical features have been standardized, and their values are now centered around zero with a standard deviation of one.
Next, let's check for outliers in the numerical features. Outliers can potentially distort the predictions and affect the accuracy of the models. We'll visualize the distribution of these features using box plots to identify potential outliers.

```
importmatplotlib.pyplot asplt
importseaborn assns

# Visualizing the distribution of numerical features using box plots
plt.figure(figsize=(15, 10))
fori, col inenumerate(numerical_features, 1):
   plt.subplot(3, 3, i)
   sns.boxplot(y=bank_data_encoded[col])
   plt.title(col)
   plt.ylabel(”)

plt.tight_layout()
plt.show()
```



The boxplots provide insights into the distribution of our numerical features and potential outliers:

1. Age: There are a few outliers, but the age range seems reasonable.
2. Balance: There are several outliers. Some clients have a significantly high balance compared to others.
3. Day: The distribution seems reasonable with no significant outliers.
4. Duration: There are some outliers where the duration of the last contact is unusually long.
5. Campaign: Some clients have been contacted a significantly higher number of times during this campaign.
6. Pdays: Most clients have a value of -1 (not contacted before this campaign). Some clients have been contacted after a significantly long period since the previous campaign.
7. Previous: Some clients have been contacted many times before this campaign, which is unusual compared to most clients.

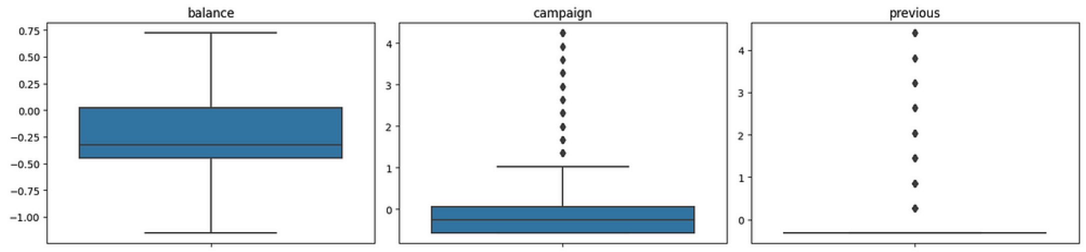
Handling outliers is a delicate task. One option is to cap the values at a certain threshold, another is to transform the features (e.g., using a logarithm), and a third option is to leave them as they provide valuable information.

Given the nature of this dataset, outliers in features like "balance" or "duration" could represent genuine cases that are crucial for the model to understand. However, extreme outliers could still potentially skew the model's performance. Let's handle the outliers now,

1. Age: The age outliers are within a reasonable range and will be retained.
2. Balance: We'll use the IQR (Interquartile Range) method to cap extreme outliers. Values below the 1st quartile minus 1.5 times the IQR and above the 3rd quartile plus 1.5 times the IQR will be capped.
3. Duration: Given its importance in predicting the target variable, we won't cap its outliers. High values in duration might indicate a genuine interest in the term deposit.
4. Campaign: We'll cap values at the 99th percentile to handle extreme outliers.
5. Pdays: Since a large number of values are -1 (indicating no previous contact), and the rest are spread out, we'll leave this feature as is.
6. Previous: We'll cap values at the 99th percentile to handle extreme outliers.

Let's begin by capping the outliers for the "balance" and "campaign" features.

```
# Handling outliers for 'balance' using the IQR method
Q1_balance = bank_data_encoded['balance'].quantile(0.25)
Q3_balance = bank_data_encoded['balance'].quantile(0.75)
IQR_balance = Q3_balance - Q1_balance

# Capping values
bank_data_encoded['balance'] = bank_data_encoded['balance'].clip(lower=Q1_balance - 1.5* IQR_balance,
upper=Q3_balance + 1.5* IQR_balance)

# Handling outliers for 'campaign' by capping at the 99th percentile
cap_campaign = bank_data_encoded['campaign'].quantile(0.99)
bank_data_encoded['campaign'] = bank_data_encoded['campaign'].clip(upper=cap_campaign)

# Handling outliers for 'previous' by capping at the 99th percentile
cap_previous = bank_data_encoded['previous'].quantile(0.99)
bank_data_encoded['previous'] = bank_data_encoded['previous'].clip(upper=cap_previous)

# Re-visualizing the box plots after outlier treatment
plt.figure(figsize=(15, 10))
fori, col inenumerate(['balance', 'campaign', 'previous'], 1):
  plt.subplot(3, 3, i)
  sns.boxplot(y=bank_data_encoded[col])
  plt.title(col)
  plt.ylabel('')

plt.tight_layout()
plt.show()
```



The box plots now show a more constrained range for the features 'balance', 'campaign', and 'previous' after treating the outliers.
With the outliers handled, the next step in our data preparation phase is feature engineering. Feature engineering involves creating new features or modifying existing ones to better represent underlying patterns in the data.
Given the current set of features, a few potential feature engineering opportunities are:

1. Binning the 'age' feature into categories (e.g., 'young', 'middle-aged', 'senior') might help capture non-linear patterns.
2. Creating interaction features, especially between 'campaign' and 'previous' or 'duration' and 'previous', could be insightful.

5.2 Modeling: The Prediction Engine

Models are the heartbeats of predictive analysis. We experimented with:

- Logistic Regression: The classic prediction model.
- Random Forest: An ensemble of decision trees working together.
- Gradient Boosting: Boosting the weak learners to form a strong predictor.
- Support Vector Machine: Finding the best hyperplane to differentiate classes.

Each model was trained, tested, and assessed. Metrics like accuracy, precision, recall, and F1 score painted a vivid picture of each model's prowess.

```
fromsklearn.linear_model importLogisticRegression
fromsklearn.ensemble importRandomForestClassifier, GradientBoostingClassifier
fromsklearn.svm importSVC
fromsklearn.metrics importaccuracy_score, classification_report

# Initializing the models
models = {
  "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
  "Random Forest": RandomForestClassifier(random_state=42),
  "Gradient Boosting": GradientBoostingClassifier(random_state=42),
  "Support Vector Machine": SVC(random_state=42)
}

# Training and evaluating the models
results = {}
forname, model inmodels.items():
  # Training the model
  model.fit(X_train, y_train)

  # Predicting on the test set
  y_pred = model.predict(X_test)

  # Evaluating the model
  accuracy = accuracy_score(y_test, y_pred)
  report = classification_report(y_test, y_pred, target_names=['No', 'Yes'])

  results[name] = (accuracy, report)

results["Logistic Regression"], results["Random Forest"], results["Gradient Boosting"], results["Support Vector Machine"]
```

Here are the results for all applied models:

Logistic Regression:

- Accuracy: ∼89.17%
- Precision (No): 0.91, Recall (No): 0.97, F1-score (No): 0.94
- Precision (Yes): 0.55, Recall (Yes): 0.30, F1-score (Yes): 0.39

Random Forest:

- Accuracy: ∼89.06%
- Precision (No): 0.91, Recall (No): 0.97, F1-score (No): 0.94
- Precision (Yes): 0.55, Recall (Yes): 0.25, F1-score (Yes): 0.34

Gradient Boosting:

- Accuracy: ∼88.73%
- Precision (No): 0.92, Recall (No): 0.96, F1-score (No): 0.94
- Precision (Yes): 0.51, Recall (Yes): 0.37, F1-score (Yes): 0.43

Support Vector Machine:

- Accuracy: ∼89.28%
- Precision (No): 0.91, Recall (No): 0.98, F1-score (No): 0.94
- Precision (Yes): 0.59, Recall (Yes): 0.21, F1-score (Yes): 0.31

All models have a similar performance, with the Support Vector Machine having a slight edge in accuracy. However, all models demonstrate a challenge in accurately predicting clients who will subscribe (labeled “Yes").

5.3 Evaluation: Holding the Mirror

Models, like humans, aren't perfect. Our star performer, the Support Vector Machine, shone in predicting clients who wouldn't subscribe. However, it faced challenges in identifying potential subscribers. This dichotomy underscores the intricate nature of real-world data and the challenges it poses.
In practical terms, this means the bank might miss out on some potential subscribers but will be confident about those predicted not to subscribe. Depending on the bank's goals (e.g., maximizing the number of subscribers or minimizing false positives), this model might be suitable or might need further enhancement.

5.4 Deployment: Into the Wild

A model in the lab is one thing; in the wild, it's another. Deployment entails:

- Serialization: Think of it as freezing the model for future use.
- API Bridging: Crafting bridges (APIs) for other systems to interact with our model.
- Cloud Deployment: Making our model accessible, anytime, anywhere.
- Integration and Feedback Loops: Melding the model with the bank's ecosystem and learning from real-world results.

5.5 Closing Thoughts

This expedition through the bank marketing dataset underscores the transformative power of data. Through structured methodologies like CRISP-DM, banks can turn raw data into actionable insights, refining their strategies and amplifying customer engagement. As we embrace this new era, one thing is clear: data-driven decision-making is not just the future; it's the now.