Advanced Programming - 2nd Assignment

Bank Marketing Dataset Classification Analysis

Josu Salinas Colina

January 5, 2026

1 Introduction

1.1 Project Overview

The objective of this project is to analyze the Bank Marketing dataset to predict whether a client will subscribe to a term deposit. The dataset consists of data from direct marketing campaigns of a Portuguese banking institution. This analysis is performed in the context of the Advanced Programming assignment, using Student ID 100562234 as the random seed.

1.2 Use of AI

1.2.1 AI-Assisted Project Scaffolding

The initial structure of this project was created using Claude Opus 4.5 (Thinking), an advanced AI assistant. The process followed these steps:

1. **Requirements Documentation:** A detailed `requirements.MD` file was first created manually, capturing all the specific requirements for the assignment, including the five main tasks, evaluation metrics, preprocessing guidelines, and general project constraints.
2. **AI-Assisted Scaffolding:** Using a conversational interface with Claude Opus 4.5, the AI was provided with the requirements document and the existing data folder structure. The explicit instruction was to create only the barebones project structure, placeholder files with proper organization and TODO markers, without implementing any actual functionality.
3. **Generated Structure:** The AI created the complete project scaffold, including:
    - Python source files (`src/`) with placeholder functions and docstrings
    - LaTeX report files (`report/`) with section templates
    - Supporting files such as `README.md` and `.gitignore`
    - Directory structure for outputs and figures

1.2.2 AI-Assisted Code review and documentation

After the code was manually developed the same conversational interface was used to ask for a review of the code, validating the changes and checking for suggestions. Finally, after the code was deemed complete and correct, the same system was used to generate code comments and descriptions of the whole project. This was particularly useful for tedious tasks such as proper formatting of outputs. AI was also used to aid in generating the code that generates the graph of the decision tree that is used in this report.

1.2.3 AI review of report

This same report was provided to the same AI assistant to validate the report and check for suggestions, the source LaTeXcode was provided in the same repository as the whole project so that the AI assistant could have access to the whole context, including the ability to read the output of the python script.

2 Task 1: Simplified Exploratory Data Analysis

This section presents the results of a simplified Exploratory Data Analysis (EDA) performed on the Bank Marketing dataset.

2.1 Dataset Overview

The Bank Marketing dataset contains information about direct marketing campaigns (phone calls) of a Portuguese banking institution. The goal is to predict whether a client will subscribe to a term deposit.

Table 1: Dataset Dimensions

Metric	**Value**
Number of instances	11,000
Number of features	16
Target variable	`deposit`

2.2 Variable Types

Variables were classified based on their data types. Columns with `object` type are considered categorical, while numeric types (`int64`, `float64`) are considered numerical.

2.2.1 Numerical Variables (7)

- `age` – age in years
- `balance` – average yearly balance
- `day` – last contact day
- `duration` – last contact duration in seconds
- `campaign` – number of contacts during this campaign
- `pdays` – days since last contact from previous campaign
- `previous` – number of contacts before this campaign

2.2.2 Categorical Variables (10)

- `job` – type of job (12 unique values)
- `marital` – marital status (3 unique values)
- `education` – education level (4 unique values)
- `default` – has credit in default (2 unique values)

- `housing` – has housing loan (2 unique values)
- `loan` – has personal loan (2 unique values)
- `contact` – contact communication type (3 unique values)
- `month` – last contact month (12 unique values)
- `poutcome` – outcome of previous campaign (4 unique values)
- `deposit` – target variable (2 unique values)

2.2.3 High Cardinality Variables

Using a threshold of 10 unique values, the following categorical variables exhibit high cardinality:

Table 2: High Cardinality Categorical Variables

Variable	Unique Values
`job`	12
`month`	12

These variables may require special encoding strategies (e.g., target encoding) in preprocessing, though standard one-hot encoding remains viable given the moderate cardinality.

2.3 Missing Values Analysis

Table 3: Columns with Missing Values

Column	Missing Count	Percentage
`job`	327	2.97%

Only the `job` column contains missing values, representing approximately 3% of the data. This will require imputation during preprocessing, where the most frequent category or a separate "missing" category can be used.

2.4 Constant and ID Columns

The analysis checked for:

- **Constant columns**: Columns where all values are identical (useless for prediction)
- **Potential ID columns**: Columns where all values are unique (likely identifiers)

Result: No constant columns and no potential ID columns were found in the dataset. All features contain meaningful variation.

2.5 Problem Type Identification

Table 4: Target Variable Analysis

Property	**Value**
Target variable	`deposit`
Data type	object (categorical)
Unique values	2 (yes, no)
Problem type	**Binary Classification**

2.6 Class Imbalance

Table 5: Class Distribution

Class	**Count**	**Percentage**
no	5,780	52.55%
yes	5,220	47.45%
Imbalance ratio	1.11	

The dataset is **not imbalanced**. With an imbalance ratio of 1.11 (using a threshold of 1.5), the classes are well-balanced. This means we can use accuracy as an appropriate evaluation metric without requiring techniques like oversampling or class weighting.

2.7 Special Variable: pdays

The `pdays` variable requires special attention as it encodes two types of information:

- **-1**: Client was not contacted in a previous campaign (or unknown)
- $\geq$ **0**: Number of days since the client was last contacted

Table 6: pdays Analysis

Metric	**Value**
Total observations	11,000
Values = -1 (no previous contact)	8,203 (74.57%)
Values > -1 (previous contact)	2,797 (25.43%)
Statistics for contacted clients only:	
Minimum	1 days
Maximum	854 days
Mean	204.72 days
Median	182 days

2.7.1 Preprocessing Alternatives

Three preprocessing strategies were considered for handling `pdays`:

1. **Option A: Median Imputation**

- Replace -1 values with the median of valid pdays values (182 days)
- *Pros*: Simple, maintains single feature
- *Cons*: Loses information about whether contact occurred; may introduce misleading patterns

2. **Option B: Large Value Replacement**
   - Replace -1 with a large value (e.g., 999 or max+1 = 855)
   - *Pros*: Simple, maintains single feature, distinguishes no-contact cases
   - *Cons*: Arbitrary choice of replacement value; may affect distance-based methods
3. **Option C: Two-Feature Approach (Selected)**
   - Create binary indicator: `was_contacted_before` = 1 if pdays $\neq$ -1, else 0
   - Keep transformed `pdays` column (with -1 replaced by 0)
   - *Pros*: Preserves maximum information; separates "whether" from "when"
   - *Cons*: Adds one feature; slightly more complex preprocessing

Option C was chosen because it preserves the maximum amount of information from the original variable. The binary flag captures whether a previous contact occurred (which is highly predictive), while the continuous value captures the recency of that contact for clients who were contacted.

3 Task 2: Outer and Inner Evaluation

3.1 Evaluation Strategy Overview

The evaluation follows a two-level approach:

1. **Outer Evaluation (Holdout):** A train/test split to estimate final model performance
2. **Inner Evaluation (Cross-Validation):** Used during hyperparameter tuning to avoid overfitting

This nested approach ensures that the test set remains truly unseen until the final evaluation.

3.2 Target Variable Encoding

The target variable `deposit` contains string values ('yes' and 'no'). To ensure compatibility with scikit-learn models (especially K-Nearest Neighbors), these were encoded into numeric values using `LabelEncoder`:

- `no` $\rightarrow$ 0
- `yes` $\rightarrow$ 1

This encoding is performed prior to the train/test split to ensure consistency across all data subsets.

3.3 Outer Evaluation: Train/Test Split

Table 7: Train/Test Split Configuration

Parameter	Value
Split ratio	80% train / 20% test
Training samples	8,800
Test samples	2,200
Stratification	Yes (on target variable)

The split uses `train_test_split` with `stratify=y` to maintain class proportions in both sets:

Table 8: Class Distribution After Split

	Training Set		Test Set	
Class	Count	%	Count	%
no	4,624	52.55%	1,156	52.55%
yes	4,176	47.45%	1,044	47.45%

3.4 Inner Evaluation Strategy

For hyperparameter tuning (Task 3), `StratifiedKFold` cross-validation is used:

Table 9: Cross-Validation Configuration

Parameter	Value
Strategy	StratifiedKFold
Number of folds	3
Shuffle	True

Justification for StratifiedKFold: While the dataset is fairly balanced, stratification ensures each fold maintains the same class proportions, leading to more reliable performance estimates during hyperparameter tuning.

3.5 Preprocessing Pipeline

All preprocessing is implemented using scikit-learn `Pipeline` and `ColumnTransformer` to ensure proper fit/transform separation and prevent data leakage.

Table 10: Preprocessing Steps by Feature Type

Feature Type	Imputation	Transformation
Numerical (6 features)	Median	StandardScaler
Categorical (10 features)	Most frequent	OneHotEncoder
pdays (special)	Custom	Binary flag + scaled continuous

3.5.1 Custom pdays Preprocessing

A custom `PdaysTransformer` was implemented to handle the `pdays` variable:

Listing 1: PdaysTransformer (Option C)

```
class PdaysTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self  # Stateless transformation

    def transform(self, X):
        was_contacted = (X != -1).astype(int)
        pdays_transformed = np.where(X == -1, 0, X)
        return np.column_stack([was_contacted,
                                pdays_transformed])
```

This creates two features from the original `pdays`:

- `was_contacted_before`: Binary indicator (1 if previously contacted)
- `pdays_transformed`: Days since contact (with -1 replaced by 0)

3.5.2 Feature Expansion

After preprocessing, the feature space expands due to one-hot encoding:

Table 11: Feature Dimensions

Stage	Dimensions
Original features	16
After preprocessing	53

3.6 Evaluation Metrics

3.6.1 Main Metric: Accuracy

Accuracy is appropriate as the main metric because:

- The classes are balanced (52.5% vs 47.5%)
- It provides an intuitive interpretation of model performance
- It is widely used for comparison in classification tasks

3.6.2 Secondary Metric: Confusion Matrices

Confusion matrices provide detailed insight into:

- True positives and true negatives
- Type I errors (false positives) and Type II errors (false negatives)
- Potential class-specific performance issues

4 Task 3: Decision Trees and KNN Comparison

This section presents the training, evaluation, and comparison of Decision Tree and K-Nearest Neighbors classifiers, along with hyperparameter optimization.

4.1 Baseline: Dummy Classifier

A Dummy classifier using the "most_frequent" strategy provides a baseline that any real model should beat. This classifier always predicts the majority class.

Table 12: Dummy Classifier Results

Metric	Value
Strategy	Most frequent
Accuracy	52.55%
Training time	0.03s

Table 13: Confusion Matrix: Dummy Classifier

	Predicted No	Predicted Yes
Actual No	1156	0
Actual Yes	1044	0

The baseline accuracy of 52.55% reflects the majority class proportion, confirming the balanced dataset analysis from Task 1.

4.2 Decision Trees

4.2.1 Default Hyperparameters

Decision Tree with scikit-learn's default parameters:

Table 14: Decision Tree (Default) Results

Metric	Value
Accuracy	79.86%
Training time	0.09s

Table 15: Confusion Matrix: Decision Tree (Default)

	Predicted No	Predicted Yes
Actual No	954	202
Actual Yes	241	803

4.2.2 Hyperparameter Tuning

GridSearchCV was used to tune the Decision Tree with the following parameter grid:

Listing 2: Decision Tree Parameter Grid

```
param_grid = {
    'max_depth': [3, 5, 10, 15, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}
```

```
# Total: 90 combinations x 3 folds = 270 fits
```

Table 16: Decision Tree (Tuned) Results

Metric	Value
Test Accuracy	82.95%
CV Mean (±std)	81.44% (±0.18%)
HPO Time	4.27s
`max_depth`	10
`min_samples_split`	5
`min_samples_leaf`	1
`criterion`	gini

Table 17: Confusion Matrix: Decision Tree (Tuned)

	Predicted No	Predicted Yes
Actual No	984	172
Actual Yes	203	841

4.3 K-Nearest Neighbors (KNN)

4.3.1 Pipeline Construction

KNN requires preprocessing through a pipeline that includes:

- Numerical features: Imputation (median) + Scaling
- Categorical features: Imputation (most frequent) + One-hot encoding
- pdays: Custom transformer (binary flag + transformed continuous)

4.3.2 Scaling Method Comparison

Two scaling methods were compared using 3-fold cross-validation:

Table 18: Scaler Comparison for KNN

Scaler	CV Accuracy	CV Std	CV Time
StandardScaler	**79.80%**	±0.67%	1.84s
MinMaxScaler	73.74%	±0.01%	1.18s

Selected scaler: StandardScaler (+6.06 percentage points over MinMaxScaler)

StandardScaler significantly outperforms MinMaxScaler for this dataset. This is likely because:

- StandardScaler is more robust to outliers in the numerical features
- The z-score normalization better preserves the relative distances important for KNN

4.3.3 Default Hyperparameters

KNN with default parameters (k=5) and StandardScaler:

Table 19: KNN (Default) Results

Metric	Value
Accuracy	81.18%
Training time	0.03s
Scaler	StandardScaler

Table 20: Confusion Matrix: KNN (Default)

	Predicted No	Predicted Yes
Actual No	985	171
Actual Yes	243	801

4.3.4 Hyperparameter Tuning

GridSearchCV was used to tune KNN:

Listing 3: KNN Parameter Grid

```
param_grid = {
    'n_neighbors': [3, 5, 7, 11, 15, 21],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}
# Total: 36 combinations x 3 folds = 108 fits
```

Table 21: KNN (Tuned) Results

Metric	Value
Test Accuracy	82.18%
CV Mean (±std)	81.06% (±0.38%)
HPO Time	11.75s
`n_neighbors`	11
`weights`	distance
`metric`	euclidean

Table 22: Confusion Matrix: KNN (Tuned)

	Predicted No	Predicted Yes
Actual No	997	159
Actual Yes	233	811

4.4 Comparison Results

Table 23: Model Comparison Summary

Model	Test Accuracy	CV Mean	Train Time
Dummy (baseline)	52.55%	–	0.03s
Decision Tree (default)	79.86%	–	0.09s
Decision Tree (tuned)	**82.95%**	81.44%	4.27s
KNN (default)	81.18%	–	0.03s
KNN (tuned)	82.18%	81.06%	11.75s

4.5 Summary

- All models significantly outperform the dummy baseline
- StandardScaler is the preferred scaler for KNN (+6.06% over MinMaxScaler)
- Hyperparameter optimization improves both models
- **Best model:** Decision Tree (tuned) at 82.95%, outperforming KNN (tuned) at 82.18%
- Decision Tree is preferred for Task 4 due to better accuracy, faster training, and interpretability

5 Task 4: Results and Final Model

This section details the selection of the best model, its evaluation on the test set, and the generation of predictions for the competition dataset.

5.1 Best Model Selection

Based on the comparison in Task 3, the **Decision Tree (Tuned)** was selected as the final model.

Table 24: Selected Model: Decision Tree (Tuned)

Metric	Value
Test Accuracy	82.95%
Training Time	4.27s
Complexity	Moderate (Depth=10)

The Decision Tree was chosen over KNN because:

1. **Higher Accuracy:** Decision Tree (82.95%) outperforms KNN (82.18%).
2. **Efficiency:** It trains and optimizes significantly faster (4.27s vs 11.75s for HPO).
3. **Inference Speed:** Decision Trees provide faster predictions than KNN, which scales linearly with dataset size.
4. **Interpretability:** The tree structure allows for easier explanation of decision rules compared to distance-based neighbors.

5.2 Outer Evaluation (Test Set)

The outer evaluation on the held-out test set (2,200 samples) confirms the model's ability to generalize:

- **Accuracy:** 82.95%
- **Performance vs Baseline:** +30.40 percentage points over the Dummy classifier (52.55%).

Table 25: Confusion Matrix: Final Model (Test Set)

	Predicted No	**Predicted Yes**
Actual No	984	172
Actual Yes	203	841

5.3 Final Model Training

To maximize performance for the competition, the final model was retrained on the full dataset (combining training and test sets).

Table 26: Final Feature Space

Parameter	**Value**
Total Instances	11,000
Original Features	16
Transformed Features	53

The final pipeline consists of:

1. **Preprocessing:** ColumnTransformer (Imputation, Scaling, Encoding, Pdays transformation).
2. **Classifier:** DecisionTreeClassifier with optimized hyperparameters:
   - `criterion='gini'`
   - `max_depth=10`
   - `min_samples_split=5`
   - `min_samples_leaf=1`

5.3.1 Decision Tree Visualization

One of the key advantages of Decision Trees is their interpretability. Figure 1 shows the top levels of the trained decision tree, illustrating how the model makes classification decisions.

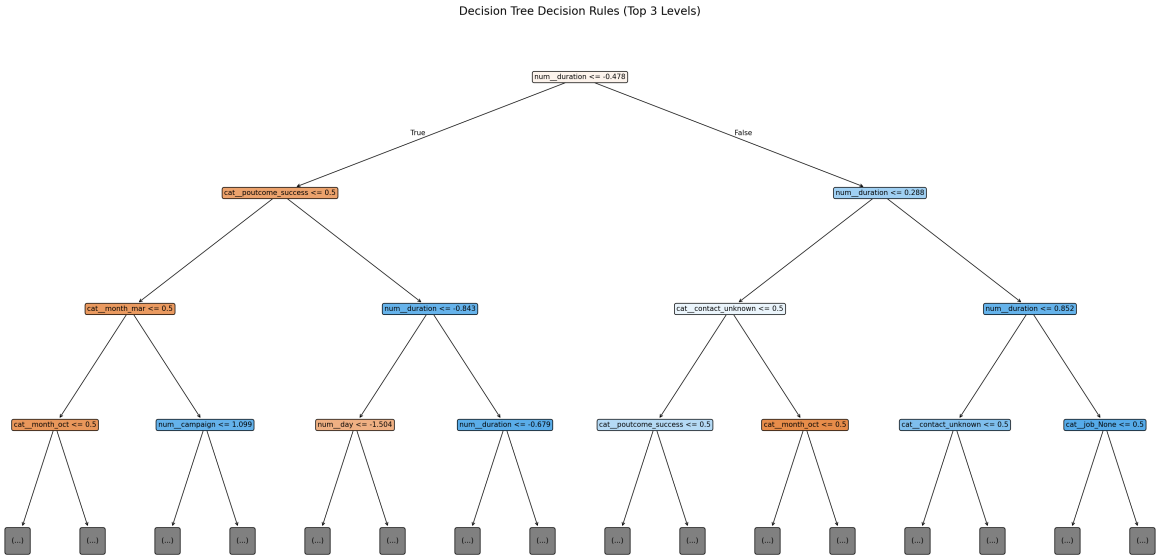

Figure 1: Decision Tree Structure (Top 3 Levels). The tree uses features like `duration` (call duration) and `poutcome` (previous campaign outcome) as primary splitting criteria. Colors indicate class predictions (orange for "no", blue for "yes").

The full tree has a depth of 10 with 245 leaf nodes, trained on 53 transformed features. The visualization demonstrates how the model can be interpreted to understand which features drive predictions.

5.4 Competition Predictions

Predictions were generated for the `bank_competition.pkl` dataset (162 instances) using the retrained final model.

- **Input:** 162 samples, 16 features.
- **Output:** 162 predictions mapped to original 'yes'/'no' labels.
- **File:** `outputs/competition_predictions.csv`

The predictions follow the required format (Id, deposit). A sample of the output is shown below:

Table 27: Sample Competition Predictions

Id	Age	Job	Marital	Balance	Pdays	Prediction
5553	43	management	married	78	109	no
915	34	housemaid	married	0	-1	yes
7652	54	technician	married	3323	-1	no
5065	43	blue-collar	single	-399	-1	no
3338	35	blue-collar	married	262	181	yes

6 Task 5: Open-Choice Task (CatBoost)

For the open-choice task, **CatBoost** was selected, a gradient boosting library developed by Yandex.

6.1 Justification

The selection of CatBoost was driven by the nature of the dataset's features:

1. **Categorical Data Handling:** The dataset contains 10 categorical variables, including `job` (12 levels) and `month` (12 levels). Standard One-Hot Encoding (used in Task 3/4) increases dimensionality significantly (from 16 to 53 features). CatBoost handles categorical features natively using techniques like Ordered Target Statistics, which often preserves more information and reduces sparsity.
2. **Gradient Boosting:** Gradient Boosted Decision Trees (GBDT) generally outperform single Decision Trees (Task 4) by iteratively correcting errors of previous trees.
3. **Robustness:** CatBoost is known for performing well with default hyperparameters, reducing the need for extensive tuning.

6.2 Implementation

Unlike the scikit-learn pipeline used in previous tasks, the implementation for CatBoost involved:

- **No One-Hot Encoding:** Categorical features were passed directly to the model as raw strings/objects.
- **Pdays Transformation:** The same 2-feature transformation logic for `pdays` was applied manually to untangle the '-1' value.
- **Missing Values:** `job` missing values were explicitly filled with 'unknown'.

The model was trained with the following configuration:

- **Iterations:** 1000 (Early stopping at 50 rounds)
- **Depth:** 6
- **Learning Rate:** 0.05
- **Loss Function:** Logloss (optimized for Accuracy)

6.3 Results Evaluation

The CatBoost model was compared against the tuned Decision Tree from Task 4.

Table 28: Task 5: CatBoost Performance vs Baseline

Metric	Decision Tree (Tuned)	CatBoost (Default)	Improvement
Accuracy	82.95%	**87.23%**	+4.28%
Training Time	4.27s	7.41s	-3.14s

Table 29: Confusion Matrix: CatBoost

	Predicted No	**Predicted Yes**
Actual No	991	165
Actual Yes	116	928

Analysis The CatBoost model achieved a significant improvement over the best model from previous tasks.

- **Accuracy:** Increased from 82.95% to 87.23%.
- **Recall (Positive Class):** The most notable improvement is in detecting successful deposits. The Decision Tree correctly identified 841 positives, while CatBoost identified 928 (an increase of 87 true positives).
- **False Negatives:** Missed opportunities dropped from 203 to 116.

6.4 Conclusion

The selection of CatBoost was highly effective. The native handling of categorical features properly exploited the information in high-cardinality variables like 'job' and 'month' without the dilution caused by massive one-hot encoding. This demonstrates that for tabular data with mixed feature types, gradient boosting with specialized categorical handling is superior to single decision tree baselines.

7 Task 6: Feature Selection for KNN (Optional)

In this optional task, it was explored whether the dimensionality of the dataset could be reduced without compromising the performance of the KNN model. The `SelectKBest` method combined with GridSearchCV was used to determine the optimal number of features.

7.1 Methodology

- **Feature Selection Method:** `SelectKBest` with the ANOVA F-value (`f_classif`) scoring function.
- **Model:** K-Nearest Neighbors Classifier with the optimal hyperparameters found in Task 3 ($k = 11$, weights='distance', metric='euclidean').
- **Search Strategy:** Grid Search over the parameter k (number of features) ranging from 5 to 53.
- **Preprocessing:** Standard One-Hot Encoding pipeline used in Task 2.

7.2 Results

The Grid Search identified the following optimal configuration:

- **Optimal Number of Features:** 53 (out of 53 total features).
- **Test Accuracy with Selected Features:** 82.18%

Analysis The feature selection process determined that all 53 features should be retained:

- **Dimensionality Reduction:** No reduction was achieved. The full feature set was identified as optimal by the grid search.
- **Performance:** The test accuracy of 82.18% is consistent with the full KNN model results from Task 3, as expected given that no features were removed.

7.3 Conclusion

The feature selection process found that all features contribute to the KNN model's performance. This indicates that the preprocessing and feature engineering steps in Task 2 did not introduce redundant or noisy features relative to the distance-based learning of KNN.