Project 2: Predicting Mortgage Default

Scarlett Jiang, Jiaying Yang

April 15, 2024

1 Machine Learning in Python - Project 2

Due Monday, April 15th by 4 pm.

Jiaying Yang & Can Jiang

1.1 Setup

[2]:
```
# Data libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from catboost import CatBoostClassifier, Pool
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import VarianceThreshold
from sklearn.ensemble import AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from sklearn.utils import class_weight as cw
import tensorflow as tf
from tensorflow.keras.metrics import Precision
from tensorflow.keras.metrics import Recall
from tensorflow.keras import backend as K
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Dropout
```

```
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTENC

# Suppress all warnings
warnings.filterwarnings("ignore")

# Plotting defaults
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80
```

[3]:
```
# Load data in easyshare.csv
d = pd.read_csv("freddiemac.csv")
```

2 Introduction

In our contemporary financial landscape, accurately forecasting mortgage defaults is crucial for financial institutions to maintain the stability and profitability of their loan portfolios. Consumers commit mortgage default when they are unable to make the loan payments as agreed [1]. The ability to predict mortgage delinquencies is paramount for institutions like Freddie Mac, enabling them to mitigate potential losses and enhance their risk management practices [2]. Effective default forecasting supports the sustainability of mortgage-backed securities and facilitates robust financial planning and strategic decision-making [3].

This project aims to develop a predictive model that can identify potential mortgage defaults with high accuracy, leveraging the available data to identify key risk factors and improve Freddie Mac's risk assessment capabilities. The data provided by Freddie Mac is being used, which comprises comprehensive loan-level credit performance information from mortgages purchased or guaranteed by Freddie Mac during the years 2017 and 2018 [4]. Moreover, in compliance with the Federal Housing Finance Agency's emphasis on risk sharing, this project also supports Freddie Mac's regulatory risk-sharing initiatives by improving risk assessment methodologies. The project results will provide critical insights into reducing the risk of mortgage default that can significantly impact future lending practices.

A significant challenge in predictive modelling of mortgage defaults is the inherent class imbalance typically present in such datasets, where defaults are significantly less frequent than non-defaults. This imbalance can lead to models biased towards predicting the majority class, resulting in many false negatives for the critical minority class (defaults). To address this challenge, our study employs the Synthetic Minority Oversampling Technique for Nominal and Continuous Data (SMOTE-NC). This method synthetically generates examples in the minority class, thereby balancing the dataset and allowing for the development of a model that more accurately predicts both classes.

By integrating various computational models—including decision trees, CatBoost classifiers, and neural networks—our approach evaluates their predictive accuracy to identify the most effective model for predicting mortgage defaults. The analysis focuses on adjusting these models with the balanced dataset provided by SMOTE-NC, enhancing their ability to detect default cases accurately.

Among the models tested, the CatBoost classifier, when adjusted with SMOTENC for class imbalance, emerged as the most effective tool, demonstrating significant predictive power, particularly

in identifying default cases and balancing accuracy. Furthermore, identifying the most influential features is intrinsic to our model's decision-making process. These include the property's zip code (`zipcode`), the Metropolitan Statistical Area code (`cd_msa`), the date of the first payment (`dt_first_pi`), the originating seller's name (`seller_name`), and the number of borrowers (`cnt_borr`), all of which are critical in assessing the likelihood of default. Such features reflect the nuanced relationship between borrower characteristics, loan specifics, and regional economic conditions.

This project contributes valuable insights into the predictive modelling of mortgage defaults. It serves as an instrumental tool for Freddie Mac, enabling the refinement of lending practices and fortifying risk assessment frameworks.

3 Exploratory Data Analysis and Feature Engineering

3.1 Exploratory Data Analysis

[3]:
```
# Print the first few rows of the data
d.head(10)
```

[3]:

	fico	dt_first_pi	flag_fthb	dt_matr	cd_msa	mi_pct	cnt_units	\
0	709.0	201703	9	204702	NaN	12	1	
1	649.0	201703	9	203202	33124.0	0	1	
2	747.0	201703	9	203702	41180.0	0	1	
3	711.0	201703	9	204702	20260.0	0	1	
4	751.0	201703	N	204702	NaN	35	1	
5	808.0	201703	9	203202	NaN	0	1	
6	634.0	201703	N	203202	NaN	0	1	
7	714.0	201703	9	204702	19804.0	0	1	
8	752.0	201703	9	203202	NaN	0	1	
9	673.0	201704	Y	204703	NaN	30	1	

	occpy_sts	cltv	dti	…	zipcode	id_loan	loan_purpose	\
0	P	84	26	…	51300	F117Q1000376	N	
1	P	52	22	…	33100	F117Q1000418	C	
2	I	43	20	…	63100	F117Q1000479	N	
3	I	80	21	…	55800	F117Q1000523	P	
4	P	95	24	…	75900	F117Q1000719	P	
5	P	49	33	…	65700	F117Q1001077	N	
6	P	55	37	…	42100	F117Q1001140	P	
7	I	80	22	…	48100	F117Q1001292	N	
8	P	63	24	…	51300	F117Q1001382	C	
9	P	95	45	…	14800	F117Q1001443	P	

	orig_loan_term	cnt_borr	seller_name	servicer_name	flag_sc	\
0	360	2	Other sellers	Other servicers	NaN	
1	180	2	Other sellers	Other servicers	NaN	
2	240	2	Other sellers	Other servicers	NaN	
3	360	2	Other sellers	Other servicers	NaN	

```
4            360        1  Other sellers  ARVESTCENTRALMTGECO     NaN
5            180        2  Other sellers      Other servicers     NaN
6            180        1  Other sellers      Other servicers     NaN
7            360        2  Other sellers      Other servicers     NaN
8            180        2  Other sellers      Other servicers     NaN
9            360        2  Other sellers      Other servicers     NaN

   prepaid default
0        0       1
1        1       0
2        1       0
3        1       0
4        1       0
5        1       0
6        1       0
7        1       0
8        1       0
9        0       1

[10 rows x 28 columns]
```

The `head` method displays the first ten rows of the original dataset, which is helpful for quickly examining the organisation of the dataset, including the names of the columns and some of the values from each column to gain an understanding of the types of data and to identify any missing values. The `default` column is the target variable, which indicates whether a mortgage has defaulted or not. The other columns contain various features that will be used to predict the target variable.

[4]:
```
# Describe the data
d.describe()
```

[4]:
```
              fico    dt_first_pi        dt_matr        cd_msa       mi_pct  \
count  6103.000000    6104.000000    6104.000000   5510.000000  6104.000000
mean    744.762740  201735.558650  204456.275885  30746.564428     5.027031
std      48.208044      44.837849     573.354730  11158.605589    10.526750
min     549.000000  201702.000000  202504.000000  10180.000000     0.000000
25%     708.000000  201705.000000  204702.000000  19740.000000     0.000000
50%     753.000000  201709.000000  204706.000000  33340.000000     0.000000
75%     786.000000  201802.000000  204711.000000  40140.000000     0.000000
max     832.000000  201901.000000  204812.000000  49740.000000    35.000000

         cnt_units         cltv          dti       orig_upb          ltv  \
count  6104.000000  6104.000000  6104.000000    6104.000000  6104.000000
mean      1.030799    72.502785    35.964613  244066.186107    72.046527
std       0.246957    20.808191    15.583786  131445.871930    20.809927
min       1.000000     7.000000     3.000000   22000.000000     7.000000
25%       1.000000    65.000000    29.000000  143000.000000    64.000000
50%       1.000000    77.000000    38.000000  218000.000000    75.000000
```

```
75%       1.000000    80.000000    43.000000  328000.000000    80.000000
max       4.000000   999.000000   999.000000  795000.000000   999.000000

            int_rt       zipcode  orig_loan_term     cnt_borr      prepaid  \
count  6104.000000   6104.000000    6104.000000  6104.000000  6104.000000
mean      4.386463  61015.612713     327.018676     1.482634     0.981488
std       0.508065  29401.664859      68.374234     0.499739     0.134806
min       2.625000    800.000000      96.000000     1.000000     0.000000
25%       4.125000  33900.000000     360.000000     1.000000     1.000000
50%       4.375000  65000.000000     360.000000     1.000000     1.000000
75%       4.750000  89800.000000     360.000000     2.000000     1.000000
max       6.125000  99900.000000     360.000000     2.000000     1.000000

           default
count  6104.000000
mean      0.018512
std       0.134806
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       1.000000
```

The `describe` method statistically summarises the numerical characteristics in a concise manner. The tool displays the number of non-null entries in each column and highlights any fields containing missing data. The mean provides information on the degree of central tendency, while the standard deviation (std) illustrates the degree of variability by showing the dispersion of the data around the mean.

The description provides critical insights into the financial and property characteristics of the loans. The average credit score (`fico`) of customers is 744.76, indicating a relatively high level with considerable variation. The average interest rate (`int_rt`) is 4.39%, with slight variation. However,loan-to-value ratios (`ltv`) shows considerable variation, suggesting that loans are provided for a wide range of properties. The data also show considerable variation in debt-to-income ratios (`tdi`), indicating different financial situations of borrowers. In addition, the dataset includes indicators for both non-foreclosure and foreclosure, which represent different credit histories and loan performance. A thorough examination of extreme values, particularly in ratios such as loan-to-value (`ltv`) and debt-to-income (`tdi`), is essential for accurate modelling and risk assessment.

The dataset's default indicator has a mean of around 0.02, indicating a significant bias towards non-default scenarios. This suggests that defaults are infrequent, accounting for only about 2% of all observations. The considerable imbalance towards non-defaults poses difficulties for predictive modelling, as conventional algorithms may over-prioritise the majority class and, therefore, struggle to anticipate the minority class of defaults effectively. To address this imbalance, it may be necessary to employ methods such as oversampling the underrepresented class, undersampling the overrepresented class, or using sophisticated modelling techniques that consider the imbalance in the class distribution. Implementing these tactics will facilitate the development of a more equitable model that can accurately distinguish the attributes of default and non-default instances, thereby

improving overall predictive effectiveness.

[5]:
```
# Get the info of the data
d.info()
```

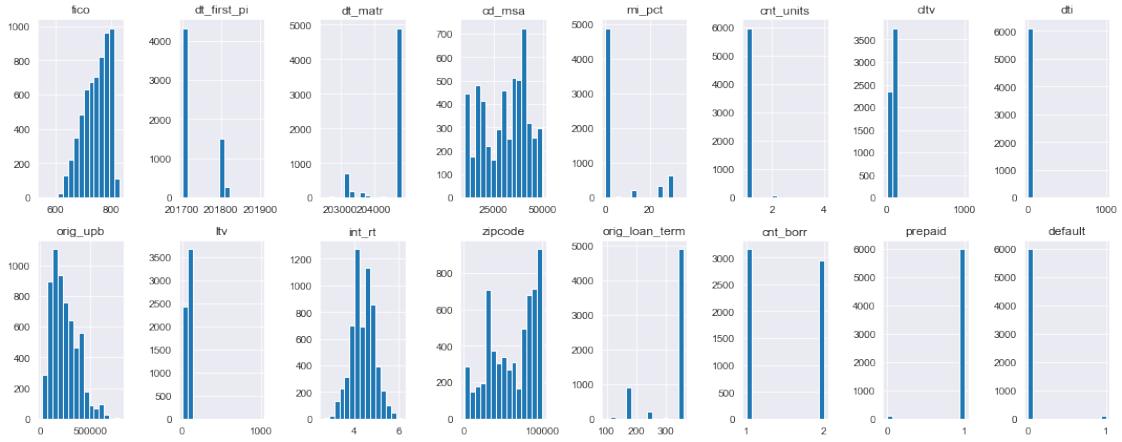
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6104 entries, 0 to 6103
Data columns (total 28 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   fico            6103 non-null   float64
 1   dt_first_pi     6104 non-null   int64
 2   flag_fthb       6104 non-null   object
 3   dt_matr         6104 non-null   int64
 4   cd_msa          5510 non-null   float64
 5   mi_pct          6104 non-null   int64
 6   cnt_units       6104 non-null   int64
 7   occpy_sts       6104 non-null   object
 8   cltv            6104 non-null   int64
 9   dti             6104 non-null   int64
 10  orig_upb        6104 non-null   int64
 11  ltv             6104 non-null   int64
 12  int_rt          6104 non-null   float64
 13  channel         6104 non-null   object
 14  ppmt_pnlty      6066 non-null   object
 15  prod_type       6104 non-null   object
 16  st              6104 non-null   object
 17  prop_type       6104 non-null   object
 18  zipcode         6104 non-null   int64
 19  id_loan         6104 non-null   object
 20  loan_purpose    6104 non-null   object
 21  orig_loan_term  6104 non-null   int64
 22  cnt_borr        6104 non-null   int64
 23  seller_name     6104 non-null   object
 24  servicer_name   6104 non-null   object
 25  flag_sc         353 non-null    object
 26  prepaid         6104 non-null   int64
 27  default         6104 non-null   int64
dtypes: float64(3), int64(13), object(12)
memory usage: 1.3+ MB
```

The `info` method returns a concise overview of the dataset, including the total number of columns, the data type of each column and the number of non-null entries in each column. The information provided is of paramount importance in understanding the structure of the dataset and identifying any missing values or data types that need to be converted prior to further analysis. Due to the mix of categorical and numerical characteristics in the dataset, appropriate pre-processing is required to ensure that the data is suitable for machine learning algorithms. The presence of missing values in certain columns, including `fico`, `cd_msa`, `ppmt_pnlty` and `flag_sc`, suggests that it may be

imperative to perform data imputation or removal prior to model training in order to fill these gaps. Determining the appropriate coding methods for categorical features and scaling techniques for numerical features also requires knowledge of the data types of the columns.

3.1.1 Distribution of variables

[6]:
```
# Distribution of numerical features
numeric_cols = d.select_dtypes(include=[np.number]).columns

# Plot the histograms of the numerical features
d[numeric_cols].hist(bins=15, figsize=(15, 6), layout=(2, -1))
plt.tight_layout()
plt.show()
```



The dataset shows a wide range of patterns in the distribution of its numerical values. The distribution of credit scores (`fico`) is skewed to the left, indicating that the majority of borrowers have high credit scores while lower scores are less common. The loan-related characteristic, unpaid balance (`orig_upb`), has distributions that are skewed to the right. This indicates the presence of high-value loans that are outliers and may need to be trimmed or adjusted for the models to achieve greater generalisation. The `default` and `prepaid` variables are binary and have a significant imbalance with a bias towards non-default scenarios, highlighting the need to employ imbalance handling strategies in predictive modelling. To optimise the performance of machine learning algorithms, it may be essential to perform feature engineering techniques such as normalisation or standardisation and address any outliers present in the data.

[7]:
```
# Assuming category_cols is a list of your categorical column names
category_cols = d.select_dtypes(include=['object']).columns

# Set the number of columns for subplots
n_cols = 2
n_rows = (len(category_cols) + 1) // n_cols
```

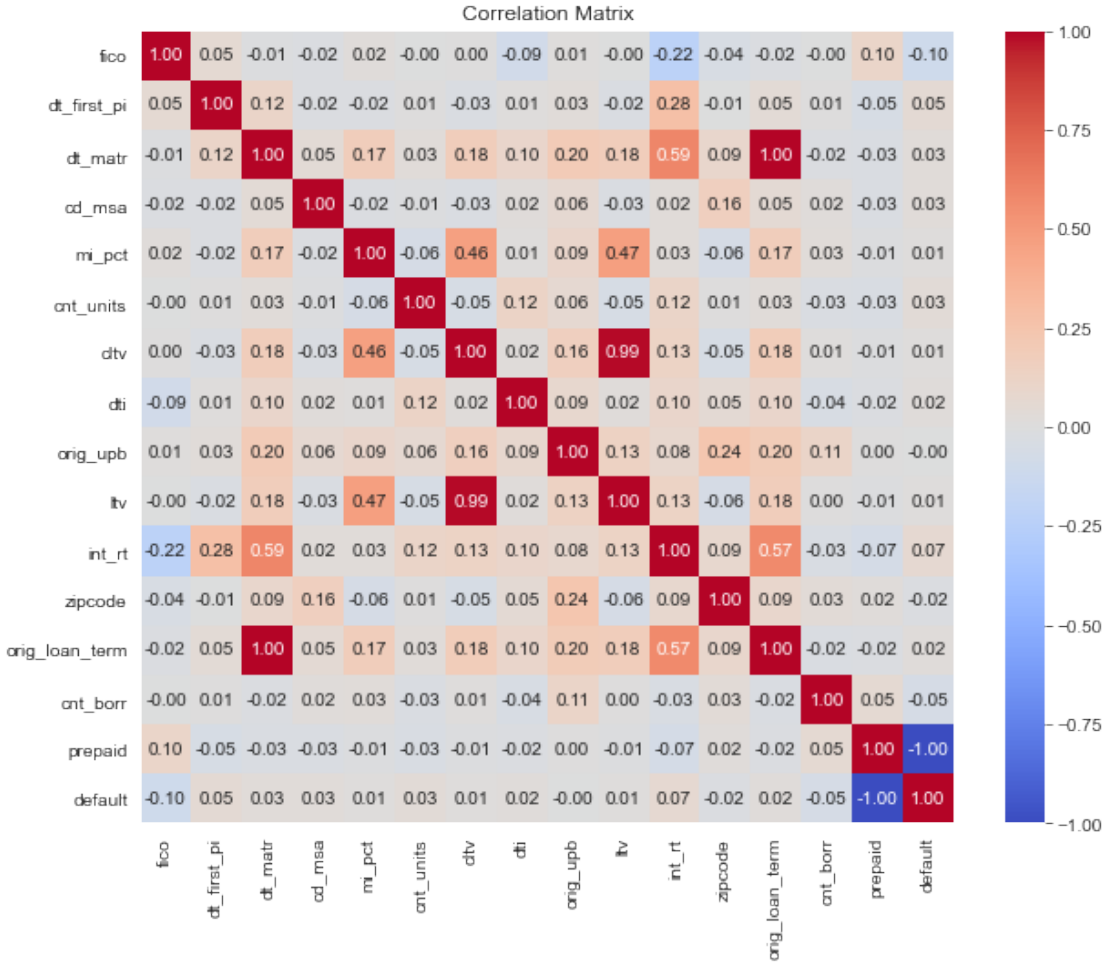
```
# Set up the matplotlib figure
plt.figure(figsize=(20, 4 * n_rows))

# Loop through the number of columns and create a countplot for each one
for i, col in enumerate(category_cols):
    plt.subplot(n_rows, n_cols, i+1)
    sns.countplot(data=d, x=col)
    plt.title(col)
    plt.xticks(rotation=45)

# Adjust layout for better display and show the plot
plt.tight_layout()
plt.show()
```

The distribution of the categorical variables shows notable disparities, with certain categories dominating others. For example, the variable `channel` reveals a predominance of a single loan origination method, suggesting that retail is the primary mode through which loans are obtained. The variable `servicer_name` also indicates a high concentration in a few services, suggesting that these companies dominate the servicing landscape. The `prop_type` variable shows heterogeneity, indicating a variety of property types in the portfolio. Significantly, the `loan_purpose` variable indicates that most loans are primarily for purchase mortagage. This observation may reflect prevailing market

trends or the specific focus of the lender. The `id_loan` variable is unique to each loan and does not provide any meaningful information for predictive modelling, which can be removed from the dataset. The presence of significant missing values in the `flag_sc` variable may require further investigation to determine the appropriate handling strategy. In summary, this uneven distribution highlights the importance of understanding market dynamics and borrower behaviour when assessing loan performance and risk. The presence of these prominent groups has the potential to affect the results of the model significantly. Feature engineering, particularly coding approaches, will play an important role in modelling. This may involve the creation of dummy variables or the use of target encoding to capture information from uneven distributions of categorical data.

3.1.2 Correlation between numerical variables

[8]:
```
# Calculate the correlation matrix
corr = d[numeric_cols].corr()

# Plot the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



The correlation heatmap shows the extent to which the features are linearly related. It is shown that the target variable, `default`, has weak correlation coefficients with most of the other features, suggesting that no single feature independently predicts default with sufficient strength. This highlights the need for comprehensive feature engineering and modelling to achieve an accurate prediction. A notable observation is the exact negative correlation (-1) between the variables `prepaid` and `default`. This suggests the two elements are mutually exclusive, as a loan cannot be prepaid and defaulted. This redundancy suggests that one of these variables could be dropped to simplify the model without losing information. The correlation matrix also reveals a strong positive correlation between the original combined loan-to-value (`cltv`) and the original loan-to-value (`ltv`) ratio, indicating that these two features are closely related. This relationship may suggest that the original combined loan-to-value ratio is a function of the original loan-to-value ratio, which could be a potential source of multicollinearity in the model. In addition, the `orig_loan_term` and `dt_matr` show a strong correlation, indicating they provide the same information and may be redundant. The presence of multicollinearity can lead to unstable model coefficients and reduced interpretability, necessitating the use of techniques such as principal component analysis (PCA) or feature selection to address this issue. The correlation matrix provides valuable insights into the relationships between features and highlights potential areas for further investigation and feature engineering to improve model performance.

3.2 Feature Engineering

[6]:
```
# Find the missing values in the data
d.isnull().sum()
```

[6]:
```
fico              1
dt_first_pi       0
flag_fthb         0
dt_matr           0
cd_msa          594
mi_pct            0
cnt_units         0
occpy_sts         0
cltv              0
dti               0
orig_upb          0
ltv               0
int_rt            0
channel           0
ppmt_pnlty       38
prod_type         0
st                0
prop_type         0
zipcode           0
id_loan           0
loan_purpose      0
```

```
orig_loan_term       0
cnt_borr             0
seller_name          0
servicer_name        0
flag_sc           5751
prepaid              0
default              0
dtype: int64
```

As we can see, there exist missing values in the columns `fico`, `cd_msa`, `ppmt_pnlty`, and `flag_sc`. These missing values need to be addressed before proceeding with the model training. Since there is only one missing value in the `fico` column, we can drop the corresponding row. For the others, we can fill the missing values with a placeholder value, such as 9, to indicate that the data is missing. This will allow us to retain the information in these columns while ensuring that the missing values do not interfere with the model training process. The `id_loan` column can be removed from the dataset as they are unique identifiers for each observation that do not provide meaningful information for the prediction. The `prepaid` column can also be removed due to its exact negative correlation with the `default` column, which suggests that the two variables are mutually exclusive. The `cd_msa` and `zipcode` columns, which were previously identified as numerical features, should be converted to object data types to ensure that they are treated as categorical features during model training. This conversion will prevent the model from interpreting these variables as ordinal or interval data, which could lead to incorrect assumptions about the relationships between the categories. The data is now ready for further processing and model training.

[4]:
```
# Drop the missing value in 'fico'
d_new = d.dropna(subset=['fico'])

# Fill the missing value in 'cd_msa' and 'flag_sc' with 9
d_new['cd_msa'] = d_new['cd_msa'].fillna(9)
d_new['ppmt_pnlty'] = d_new['ppmt_pnlty'].fillna(9)
d_new['flag_sc'] = d_new['flag_sc'].fillna(9)
d_new= d_new.drop('prepaid', axis=1)
d_new= d_new.drop('id_loan', axis=1)

# Convert 'cd_msa' and 'zipcode' columns to 'object' data type
d_new['cd_msa'] = d_new['cd_msa'].astype('object')
d_new['zipcode'] = d_new['zipcode'].astype('object')
```

Now there are no missing values in the dataset, which is ready for further processing. We split the data into training and testing sets in the proportion of 80% and 20%, respectively, to prepare for model training and evaluation.

3.2.1 Split the data into training and testing sets

[6]:
```
# List of numerical features
numeric_features = d_new.select_dtypes(include=['int64', 'float64']).
  ↪drop(['default'], axis=1).columns
```

```
# Define the target variable
y = d_new['default']
X= d_new.drop('default', axis=1)
categorical_features = X.select_dtypes(include=['object']).columns
print(categorical_features)

# Define class labels
class_labels = ['Class 0', 'Class 1']  # Replace with your actual class labels␣
 ↪if available

# Convert categorical features to string
for col in categorical_features:
    X[col] = X[col].astype(str)
```

```
Index(['flag_fthb', 'cd_msa', 'occpy_sts', 'channel', 'ppmt_pnlty',
       'prod_type', 'st', 'prop_type', 'zipcode', 'loan_purpose',
       'seller_name', 'servicer_name', 'flag_sc'],
      dtype='object')
```

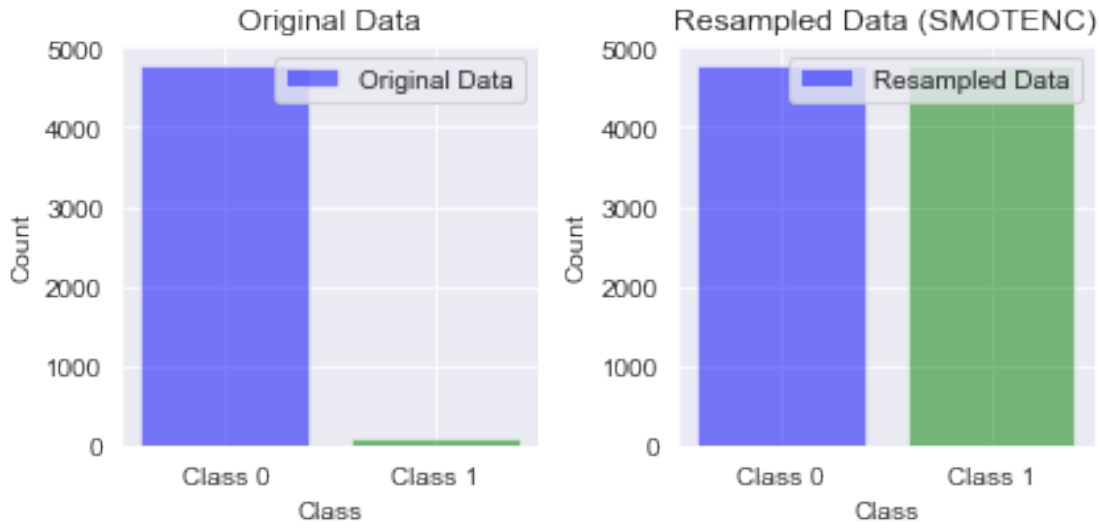
[7]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

3.2.2 Handling Class Imbalance with SMOTENC

Addressing the data imbalance is essential in this project due to the much lower incidence of loan defaults (class 1) compared to non-defaults (class 0). Failure to do so can result in biased model predictions favouring the majority class. An unbalanced dataset can cause the model to ignore the more nuanced patterns associated with the minority class, leading to inadequate predictive performance for the desired outcomes. To address this issue, we use SMOTE-NC (Synthetic Minority Over-sampling Technique for Nominal and Continuous Data), an innovative technique for increasing the representation of the minority class by generating synthetic samples. SMOTE-NC is particularly suited to datasets containing a combination of nominal (categorical) and continuous features. It is able to handle the nuances of multiple feature types by generating synthetic samples in a more sophisticated manner while preserving the original distributions. SMOTE-NC enhances the ability of models to learn patterns that are more broadly applicable by balancing the distribution of classes. This results in improved recall without a significant loss of precision, which is critical for successful risk management in lending [5]. The SMOTE-NC oversampling is used to generate novel, authentic instances of the under-represented default conditions, thereby expanding the dataset while preserving valuable information. By using this method, we not only mitigate the bias towards non-defaults but also enrich the dataset, thereby improving the models' ability to detect the complex patterns of defaults.

[8]:
```
# Apply SMOTENC for handling class imbalance
categorical_features_indices = [X_train.columns.get_loc(col) for col in␣
 ↪categorical_features]
```

```
smote_nc = SMOTENC(categorical_features=categorical_features_indices,␣
 ↪random_state=42, k_neighbors=5)

# Resample the training data using SMOTENC
X_train_resampled, y_train_resampled = smote_nc.fit_resample(X_train, y_train)

# Calculate the number of samples for each class in the original dataset
unique_classes_train, counts_train = np.unique(y_train, return_counts=True)
class_names_train = ['Class ' + str(cls) for cls in unique_classes_train]

# Calculate the number of samples for each class in the resampled dataset
unique_classes_resampled, counts_resampled = np.unique(y_train_resampled,␣
 ↪return_counts=True)
class_names_resampled = ['Class ' + str(cls) for cls in␣
 ↪unique_classes_resampled]

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(6, 3))

# Plot the first subplot: distribution of classes in the original dataset
colors_train = ['blue' if cls == 0 else 'green' for cls in␣
 ↪unique_classes_train]  # Select colors based on classes
axes[0].bar(class_names_train, counts_train, color=colors_train, alpha=0.5,␣
 ↪label='Original Data')
axes[0].set_title('Original Data')
axes[0].set_xlabel('Class')
axes[0].set_ylabel('Count')
axes[0].legend()

# Plot the second subplot: distribution of classes after SMOTENC processing
axes[1].bar(class_names_resampled, counts_resampled, color=colors_train,␣
 ↪alpha=0.5, label='Resampled Data')
axes[1].set_title('Resampled Data (SMOTENC)')
axes[1].set_xlabel('Class')
axes[1].set_ylabel('Count')
axes[1].legend()

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



After the SMOTENC oversampling, the script calculates and displays the distribution of classes before and after the application of SMOTENC using bar charts. These visualizations provide a clear before-and-after comparison, illustrating the effectiveness of SMOTENC in balancing class distribution. The dataset is now balanced, with an equal number of instances for each class. This balanced dataset will be used to train the machine learning models, ensuring that the models are not biased towards the majority class and can effectively distinguish between default and non-default scenarios. In addition, the categorical features have been transformed using one-hot encoding to ensure that the models can interpret them correctly. The numerical features have been standardised to ensure that they are on the same scale and to prevent any one feature from dominating the others. The data is now ready for the implementation of machine learning models to predict mortgage defaults.

3.2.3 Tranformation of Features after SMOTENC using OneHotEncoder and StandardScaler

[9]:
```
# Define ColumnTransformer
ct = ColumnTransformer([
    ("onehot", OneHotEncoder(handle_unknown='ignore', sparse=False),␣
 ↪categorical_features)],
    remainder='passthrough')

# Use Pipeline to concatenate OneHotEncoder and StandardScaler
pipeline = Pipeline([
    ('preprocessor', ct),
    ('scaler', StandardScaler())
])

# Process the training set data
X_train_processed = pipeline.fit_transform(X_train_resampled)
```

```
X_train_processed = pd.DataFrame(X_train_processed, columns=pipeline.
 ↪named_steps['preprocessor'].get_feature_names_out())

# Process the original training set data
X_train_origin = pipeline.transform(X_train)
X_train_origin = pd.DataFrame(X_train_origin, columns=pipeline.
 ↪named_steps['preprocessor'].get_feature_names_out())

# Process the test set data
X_test_processed = pipeline.transform(X_test)
X_test_processed = pd.DataFrame(X_test_processed, columns=pipeline.
 ↪named_steps['preprocessor'].get_feature_names_out())
```

We sets up a preprocessing pipeline using scikit-learn's ColumnTransformer and Pipeline to handle datasets with mixed categorical and continuous features. It employs OneHotEncoder to encode categorical features and ignores any unknown categories during transformation. A StandardScaler is integrated to standardize all features, ensuring uniformity in scale. The pipeline processes the training, resampled training, and test data, transforming and scaling them before converting back to pandas dataframes with appropriately named columns.

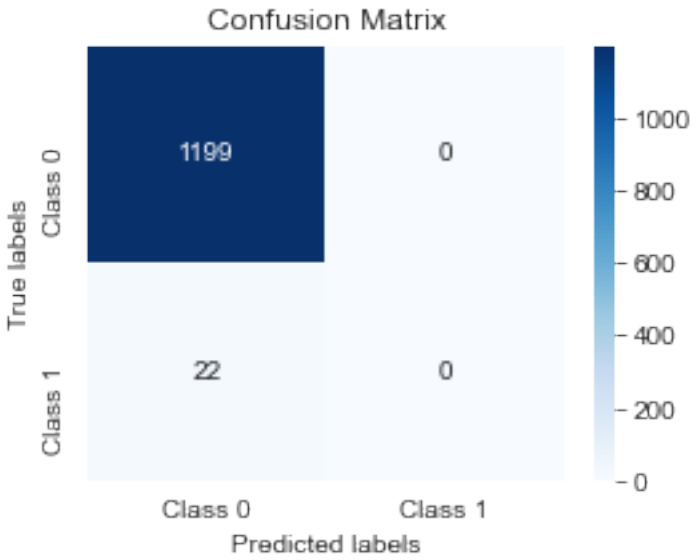
4 Model Fitting and Tuning

4.1 Baseline Model – Catboost without SMOTENC

[21]:
```
# Create an instance of CatBoostClassifier
catboost_model_baseline = CatBoostClassifier(
    iterations=100,
    learning_rate=0.2,
    depth=16,
    cat_features=categorical_features_indices,
    random_state=42,
    verbose=0
)

# Train the classifier
catboost_model_baseline.fit(X_train, y_train)

# Predict on the test data
y_pred_catboost_baseline = catboost_model_baseline.predict(X_test)

# Generate the classification report
report_catboost_processed = classification_report(y_test,␣
 ↪y_pred_catboost_baseline)

# Print the classification report
print("Classification Report:")
print(report_catboost_processed)
```

```
# Calculate the confusion matrix
conf_matrix_catboost_processed = confusion_matrix(y_test,␣
 ↪y_pred_catboost_baseline)

# Plot the confusion matrix
plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix_catboost_processed, annot=True, fmt="d", cmap="Blues",␣
 ↪xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1199
           1       0.00      0.00      0.00        22

    accuracy                           0.98      1221
   macro avg       0.49      0.50      0.50      1221
weighted avg       0.96      0.98      0.97      1221
```



CatBoost is a high-performance, open-source gradient boosting library, particularly well-suited for categorical data. It handles categorical features by using a special algorithm to transform them into numerical data, which avoids the need for extensive preprocessing and allows the model to learn more complex data patterns.

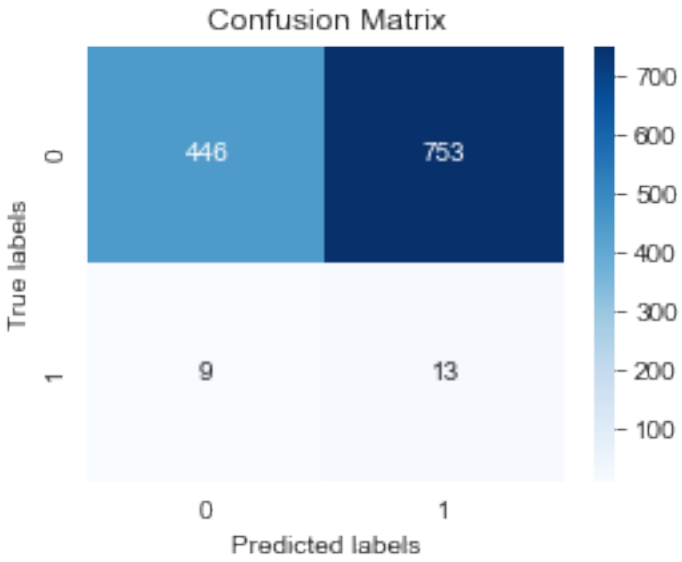
The provided result shows a perfect recall for class 0, meaning the model has successfully identified all instances of the majority class. However, for class 1, the recall, precision, and F1-score are 0, indicating the model failed to correctly identify any instances of the minority class. This is also evident in the confusion matrix, where all instances of class 1 are misclassified as class 0. Despite the high accuracy, the model's inability to recognize class 1 instances makes it an unreliable predictor for the minority class.

4.2 Gaussian Naive Bayes with SMOTENC

[20]:
```
# Initialize the GaussianNB classifier
nb_classifier = GaussianNB()

# Set the values for the var_smoothing parameter to be tested
param_grid = {'var_smoothing': np.logspace(0,-2, num=10)}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=nb_classifier, param_grid=param_grid,␣
 ↪cv=5, scoring='recall')

# Train the model
grid_search.fit(X_train_processed, y_train_resampled)

# Best parameters and performance of the best model
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))

best_model = grid_search.best_estimator_

# Predict using the best model
y_pred_nb_smotenc = best_model.predict(X_test_processed)

# Generate and print the classification report
report_smotenc_nb = classification_report(y_test, y_pred_nb_smotenc)
print("Classification Report:")
print(report_smotenc_nb)

# Calculate and plot the confusion matrix
conf_matrix_nb_smotenc = confusion_matrix(y_test, y_pred_nb_smotenc)
plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix_nb_smotenc, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

```
Best parameters: {'var_smoothing': 1.0}
Best cross-validation score: 1.00
Classification Report:
```

	precision	recall	f1-score	support
0	0.98	0.37	0.54	1199
1	0.02	0.59	0.03	22
accuracy			0.38	1221
macro avg	0.50	0.48	0.29	1221
weighted avg	0.96	0.38	0.53	1221



Despite the Gaussian Naive Bayes model showing a reasonable recall for class 1, its overall performance isn't adequate for selection due to a combination of factors. The low precision for class 1 results in a large number of false positives, which is problematic in many contexts. The model is also evidently biased towards the majority class, as seen in the confusion matrix, leading to many false negatives for class 1. This is further supported by the extremely low F1-score for class 1, indicating poor predictive reliability. Such performance suggests that the model is not effectively distinguishing between classes, and the high cross-validation score could be misleading because of the class imbalance. These issues, along with the potential violation of the Naive Bayes assumption of feature independence, make this model a less suitable choice for situations where accurate and reliable classification of the minority class is critical.

4.3 Neural Network with SMOTENC

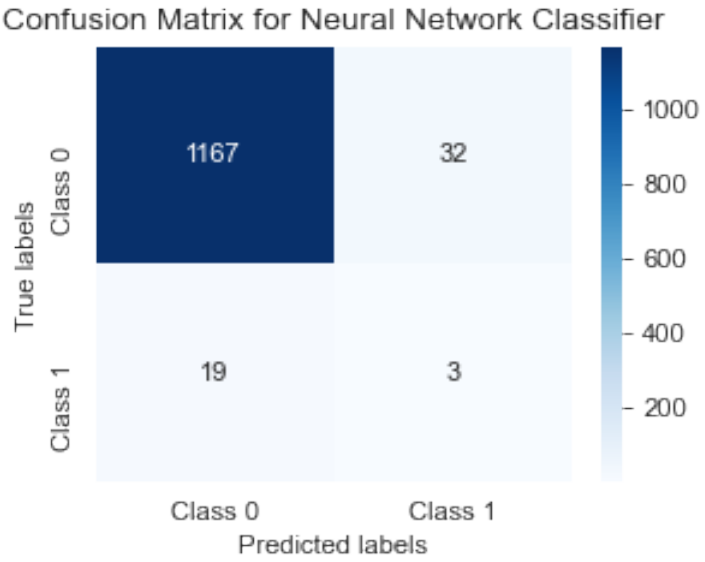
[68]:
```
def weighted_binary_crossentropy(y_true, y_pred):
    # Calculate binary cross-entropy loss
    bce = K.binary_crossentropy(y_true, y_pred)
    # Get the weight for class 1
    class_weight_for_1 = 10.0  # Example weight, adjust as needed
    # Calculate the weight vector, applying a larger weight to class 1
```

```
    weight_vector = y_true * class_weight_for_1 + (1. - y_true)
    weighted_bce = weight_vector * bce
    return K.mean(weighted_bce)

# Calculate class weights which will be used during training
class_weights = cw.compute_class_weight('balanced', classes=np.
 ↪unique(y_train_resampled), y=y_train_resampled)
class_weights_dict = {i: weight for i, weight in enumerate(class_weights)}

# Neural networks expect one-hot encoded outputs for multiclass classification
y_train_smote_one_hot = to_categorical(y_train_resampled)
y_test_hv_one_hot = to_categorical(y_test)

# Define a precision metric for class 1
precision_class_1 = Precision(class_id=1)
recall_class_1 = Recall(class_id=1)

nn_classifier = Sequential([
    Dense(128, activation='swish', input_shape=(X_train_processed.shape[1],)),
    Dropout(0.5),
    Dense(256, activation='swish'),
    Dropout(0.5),
    Dense(512, activation='swish'),
    Dropout(0.5),
    Dense(512, activation='swish'),
    Dropout(0.5),
    Dense(512, activation='swish'),
    Dropout(0.5),
    Dense(256, activation='swish'),
    Dropout(0.5),
    Dense(128, activation='swish'),
    Dropout(0.5),
    Dense(y_train_smote_one_hot.shape[1], activation='softmax')  # Assuming␣
 ↪this is for a multi-class classification problem
])

# Compile the neural network with the custom precision metric for class 1
nn_classifier.compile(optimizer='adam', loss=weighted_binary_crossentropy,␣
 ↪metrics=[recall_class_1])
nn_classifier.fit(X_train_processed, y_train_smote_one_hot, epochs=10,␣
 ↪batch_size=64)
# Predict on the testing data
y_pred_nn_smotenc = nn_classifier.predict(X_test_processed)
y_pred_nn_smotenc = y_pred_nn_smotenc.argmax(axis=1)  # Convert probabilities␣
 ↪to class labels

# Generate the classification report
```

```
report_smote_nn = classification_report(y_test, y_pred_nn_smotenc)

# Print the classification report
print("Classification Report:")
print(report_smote_nn)

# Calculate the confusion matrix
conf_matrix_nn = confusion_matrix(y_test, y_pred_nn_smotenc)

# Plot the confusion matrix
plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix_nn, annot=True, fmt="d", cmap="Blues",␣
 ↪xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for Neural Network Classifier')
plt.show()
```

```
Epoch 1/10
150/150 [==============================] - 4s 10ms/step - loss: 0.8581 - recall:
0.8708
Epoch 2/10
150/150 [==============================] - 2s 10ms/step - loss: 0.2792 - recall:
0.9827
Epoch 3/10
150/150 [==============================] - 1s 10ms/step - loss: 0.2308 - recall:
0.9831
Epoch 4/10
150/150 [==============================] - 1s 10ms/step - loss: 0.2041 - recall:
0.9856
Epoch 5/10
150/150 [==============================] - 1s 10ms/step - loss: 0.1647 - recall:
0.9833
Epoch 6/10
150/150 [==============================] - 2s 10ms/step - loss: 0.1312 - recall:
0.9862
Epoch 7/10
150/150 [==============================] - 2s 10ms/step - loss: 0.1371 - recall:
0.9850
Epoch 8/10
150/150 [==============================] - 1s 10ms/step - loss: 0.1176 - recall:
0.9885
Epoch 9/10
150/150 [==============================] - 1s 9ms/step - loss: 0.1107 - recall:
0.9894
Epoch 10/10
150/150 [==============================] - 1s 10ms/step - loss: 0.1161 - recall:
```

```
0.9885
39/39 [==============================] - 0s 2ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      1199
           1       0.09      0.14      0.11        22

    accuracy                           0.96      1221
   macro avg       0.53      0.55      0.54      1221
weighted avg       0.97      0.96      0.96      1221
```



The neural network model exhibits high recall for the majority class (class 0), but its performance on the minority class (class 1) is not satisfactory. While the recall for class 1 shows the model can identify some of the positive cases, the low precision indicates a high number of false positives. The overall accuracy is skewed by the model's performance on the majority class, which is common in imbalanced datasets. Despite the weighted loss function aimed at addressing class imbalance, the model still underperforms for the minority class,

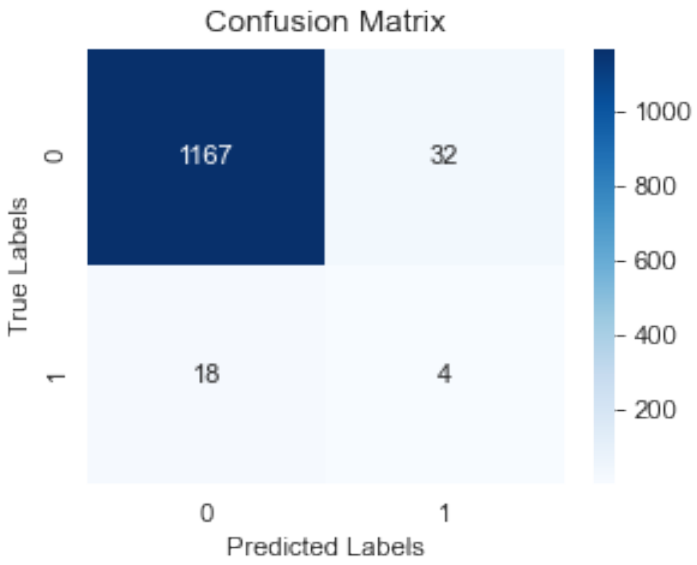
4.4 Logistic Regression with SMOTENC

4.4.1 GrdiresearchCV for Logistic Regression with accuracy

[71]:
```
# Define the model parameter grid
param_grid = {
    'solver': ['lbfgs', 'newton-cg', 'sag', 'saga'],
    'C': np.logspace(-4, 1, 10),
    'penalty': [ 'l2']
}
```

```
# Initialize the LogisticRegression classifier
logistic_classifier = LogisticRegression(random_state=42,␣
 ↪class_weight='balanced', max_iter=100)

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=logistic_classifier,␣
 ↪param_grid=param_grid, cv=5, scoring='accuracy', verbose=1)

# Perform grid search using the resampled training data
grid_search.fit(X_train_processed, y_train_resampled)

# Output the best parameters and best score
print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))

# Use the best model with the best parameters for prediction
best_model = grid_search.best_estimator_
y_pred_smotenc_logistic = best_model.predict(X_test_processed)

# Generate and print the classification report
report_smotenc_logistic = classification_report(y_test, y_pred_smotenc_logistic)
print("Classification Report:")
print(report_smotenc_logistic)

# Calculate and plot the confusion matrix
conf_matrix_logistic_smotenc = confusion_matrix(y_test, y_pred_smotenc_logistic)
plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix_logistic_smotenc, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
Best parameters found:  {'C': 10.0, 'penalty': 'l2', 'solver': 'newton-cg'}
Best cross-validation score: 0.97
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      1199
           1       0.11      0.18      0.14        22

    accuracy                           0.96      1221
   macro avg       0.55      0.58      0.56      1221
weighted avg       0.97      0.96      0.96      1221
```



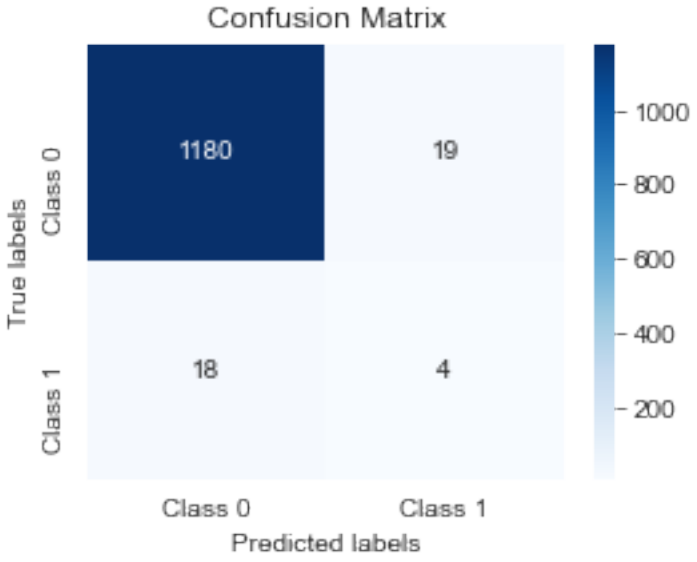
4.5 Final Model – Catboost with SMOTENC

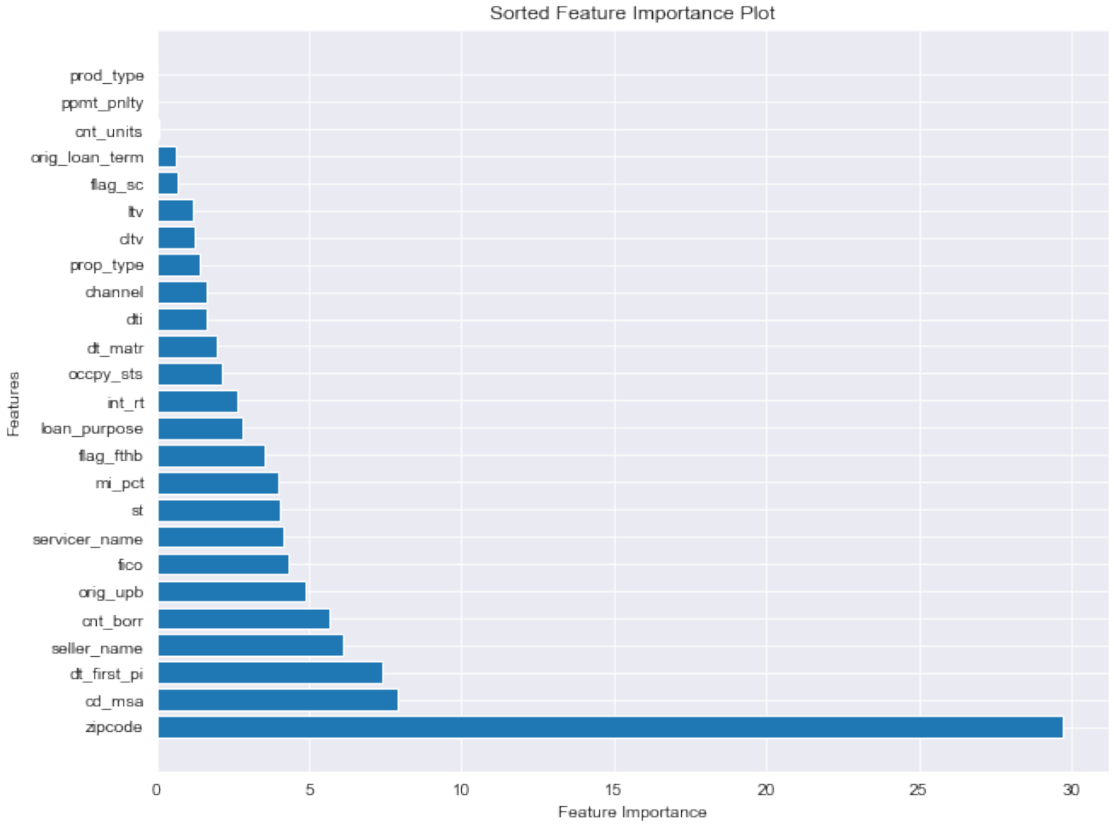
[19]:
```
# Define the parameter grid
param_grid = {
    'learning_rate': [0.1, 0.2, 0.3],  # Default: 0.2
    'depth': [ 10, 16,26],  # Default: 16
    'l2_leaf_reg': [1, 3, 5]  # Default: 3
}

# Initialize CatBoost classifier
catboost_model = CatBoostClassifier(
    iterations=30,
    cat_features=categorical_features_indices,
    random_state=42,
    verbose=0
)

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=catboost_model, param_grid=param_grid,␣
 ↪cv=2, scoring='recall', verbose=1)

# Train the model with grid search
grid_search.fit(X_train_resampled, y_train_resampled)

# Get the best model
best_model = grid_search.best_estimator_
```

```
# Predict on the test set
y_pred_catboost_gridsearch = best_model.predict(X_test)

# Generate the classification report
report_catboost_gridsearch = classification_report(y_test,␣
 ↪y_pred_catboost_gridsearch)

# Print the classification report
print("Classification Report:")
print(report_catboost_gridsearch)

# Calculate the confusion matrix
conf_matrix_catboost_gridsearch = confusion_matrix(y_test,␣
 ↪y_pred_catboost_gridsearch)

# Plot the confusion matrix
plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix_catboost_gridsearch, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Get feature importance using default "FeatureImportance" type
feature_importance = conf_matrix_catboost_gridsearch.
 ↪get_feature_importance(type='FeatureImportance')
# Sort feature importance in descending order
sorted_indices = np.argsort(feature_importance)[::-1]
sorted_features = X_train_resampled.columns[sorted_indices]
sorted_importance = feature_importance[sorted_indices]

# Plot sorted feature importance
plt.figure(figsize=(10, 8))
plt.barh(sorted_features, sorted_importance)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Sorted Feature Importance Plot')
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1199
           1       0.17      0.18      0.18        22

    accuracy                           0.97      1221
   macro avg       0.58      0.58      0.58      1221
```

```
weighted avg       0.97      0.97      0.97      1221
```





The CatBoost model, after SMOTENC resampling, shows an improved ability to identify the minority class, with an increase in recall and F1-score for class 1. Despite these improvements, the model still struggles with low precision for class 1, leading to a considerable number of false positives. This outcome suggests that while SMOTENC has made the model more sensitive to the minority class, further tuning or different modeling strategies may be necessary to achieve a better balance between precision and recall.

Although the model's precision and recall for class 1 are modest, the increased recall after applying SMOTENC reflects a better detection rate of actual defaults, which is essential for risk management. The improvement, albeit slight, demonstrates the model's enhanced sensitivity to identifying potential defaults, a critical aspect of financial modeling. The balance between interpretability and predictive power makes this model a practical choice for real-world applications where understanding the influence of specific variables is as important as prediction accuracy.

The feature importance plot indicates that loan default risk is intricately associated with geographic, economic, temporal, and institutional factors. The significance of the Metropolitan Statistical Area code and zip code underscores the impact of local and regional economic conditions on loan performance. The first payment date highlights how loan terms and broader economic circumstances at the time of loan initiation can influence default risk. The prominence of the seller's name suggests that the originator's underwriting standards and practices play a crucial role in the likelihood of a borrower defaulting. These insights reflect the complex interplay of factors that determine loan defaults and the necessity for lenders to adopt a comprehensive approach to risk assessment.

5 Discussion & Conclusions

This project aimed to develop and refine a predictive model to identify potential mortgage defaults accurately. After an extensive analysis, the final model deployed is the CatBoost classifier, bolstered by SMOTE-NC, which has clearly outperformed other similar methods. Our extensive comparative research showed that this approach performed exceptionally well regarding both recall and accuracy metrics when applied to our dataset, which consisted of a diverse and complex set of financial variables.

A crucial evaluation in mortgage default prediction models is to prioritise recall, which is important in the fields of machine learning and financial risk management. Recall, or sensitivity, is pivotal as it captures the majority of actual defaults, directly impacting a financial institution's risk exposure and financial health. Although achieving a high level of accuracy is crucial to maintaining the reliability of a model, it makes more financial sense to prioritise recall. This is because the potential consequential cost of failing to detect a default (false negatives) generally outweighs the inconvenience caused by false positives. Therefore, achieving a balance between recall and accuracy is of paramount importance in order to avoid flooding the system with false alarms and to ensure optimal operational efficiency. The model we chose, the CatBoost classifier enhanced with SMOTE-NC, effectively achieves this balance. As a result, financial institutions are provided with a reliable and effective tool to monitor and mitigate the potential for mortgage defaults proactively.

The CatBoost classifier showed a significant improvement in recall for class 1, which corresponds to the minority class representing defaults. Although models such as Gaussian Naive Bayes showed higher recall rates for Class 1, they achieved this by generating an unacceptably high number of false positives, as evidenced by their poorer accuracy scores. On the other hand, the neural network, although showing potential in some aspects, failed to achieve a satisfactory balance, leaning

too much towards the majority class and resulting in suboptimal recall for class 1. CatBoost's exceptional ability to effectively handle categorical variables, such as postcodes and MSA codes, makes it the preferred choice for our dataset [6].

SMOTE-NC played a pivotal role in improving CatBoost's performance. By generating synthetic data points it increased the representation of the underrepresented default class, thus improving the training set. This strategy was particularly appropriate for our dataset, which initially had a significant class imbalance—a typical challenge in default prediction scenarios. By correcting for this bias, SMOTE-NC enabled the CatBoost model to acquire more refined discrimination between defaults and non-defaults, thereby increasing its predictive sensitivity to defaults without compromising the overall accuracy of the model. In essence, combining the CatBoost classifier with SMOTE-NC not only balanced our dataset but also improved the accuracy and sensitivity of the model in predicting mortgage defaults and their real-world impact. It proved to be the most effective tool in our collection, providing a reliable signal of potential defaults that is essential for proactive risk management.

Within the complex realm of mortgage default prediction, our final model has identified a collection of crucial characteristics that help to understand the probability of default. These characteristics provide valuable insights into a range of economic, temporal and institutional elements, thereby illuminating not only the specific circumstances of the borrower but also more general economic indicators.

1. **Property's Zip Code**: The zipcode of a property can serve as an indicator of the socio-economic level of an area, which may be associated with the likelihood of mortgage defaults. Areas of reduced economic activity may have higher default rates due to employment volatility or a downturn in the housing market [7]. **Recommendation**: Government authorities could consider increasing the allocation of resources to economic development in postcodes with higher default rates. Alternatively, they could implement specific relief programmes targeted at homeowners in these areas.

2. **Metropolitan Statistical Area (MSA) Code**: Similar to the postcode, the MSA code provides an insight into the economic conditions of a particular region. An MSA with a strong and resilient economy is likely to have lower delinquency rates due to improved job prospects and increased property values. **Recommendation**: Identifying MSAs with elevated default risks can provide valuable insights for implementing targeted regional policies, such as job creation programmes or housing market stabilisation measures.

3. **Date of First Payment**: The timing of the first payment may indicate the borrower's financial condition at the start of the loan. Loans made during periods of economic downturn may carry a higher level of risk. **Recommendation**: To prevent defaults, lenders should offer borrowers more flexible initial payment terms or provide counselling for those whose initial payments coincide with economic downturns.

4. **Originating Seller's Name**: The originator of the mortgage may be able to indicate the underwriting standards used. Sellers with a higher default rate may have less stringent underwriting standards. **Recommendation**: Monitoring organisations that have a high number of borrowers who fail to repay their loans could lead to improved lending criteria and procedures across the sector.

5. **Number of Borrowers (cnt_borr)**: Individual borrowers may have a more significant financial burden than borrowers in a group, affecting their ability to meet their mortgage

obligations. **Recommendation**: Providing financial planning education programmes for individuals interested in becoming single homeowners or offering more favourable loan terms could help reduce the likelihood of loan default.

By using a model that incorporates these key features, lending institutions can improve the accuracy of default risk prediction and make a valuable contribution to the advancement of data-driven intervention strategies specifically designed to avert defaults.

Although our final model demonstrates resilience in numerous facets of default prediction, it has specific drawbacks that need to be recognised to provide a thorough evaluation. While the recall metric is favoured for its ability to identify defaults, it is not flawless and may result in a certain percentage of false positives. If this percentage is significant, it could compromise the accuracy of the assessment and lead to inefficiencies in the risk management process. The effectiveness of the model is also dependent on the continued applicability and accuracy of the variables used; changes in the economy and demographics may alter the significance of characteristics such as MSA codes and postcodes, which may require reassessment and adjustment of the model. It should also be noted that while the selected variables provide valuable insights, they do not capture the full range of an individual's financial circumstances or the broader macroeconomic circumstances that could potentially affect default rates. This underscores the need for the model to be integrated into a dynamic analytical framework incorporating regular updates and enhancements to maintain its predictive power and operational effectiveness in the ever-changing domain of financial risk assessment.

In conclusion, the results of this predictive modelling exercise provide a comprehensive understanding of the indicators and risks associated with mortgage default. The suggestions will enable financial institutions and policymakers to reduce the risk of mortgage default, thereby contributing to the financial well-being and stability of lending institutions.

6 References

1. Belesky, Jacqueline . "Mortgage Default: What Are the Biggest Causes?" *Canstar*, 8 Sept. 2021, www.canstar.com.au/home-loans/biggest-causes-of-mortgage-default/. Accessed 8 Apr. 2024.

2. Federal Housing Finance Agency. *2021 Report to Congress*. Federal Housing Finance Agency, 16 June 2022.

3. Kau, James B., et al. "A Generalized Valuation Model for Fixed-Rate Residential Mortgages." *Journal of Money, Credit and Banking*, vol. 24, no. 3, Aug. 1992, p. 279, https://doi.org/10.2307/1992718. Accessed 27 Apr. 2022.

4. Freddie Mac. "Single Family Loan-Level Dataset." *Www.freddiemac.com*, www.freddiemac.com/research/datasets/sf-loanlevel-dataset. Accessed 8 Apr. 2024.

5. Chawla, N. V., et al. "SMOTE: Synthetic Minority Over-Sampling Technique." *Journal of Artificial Intelligence Research*, vol. 16, no. 16, 1 June 2002, pp. 321–357, www.jair.org/index.php/jair/article/view/10302, https://doi.org/10.1613/jair.953.

6. Prokhorenkova, Liudmila, et al. "CatBoost: Unbiased Boosting with Categorical Features." *arXiv.Org*, 2019, https://doi.org/10.48550/arxiv.1706.09516.

7. Mian, Atif, and Amir Sufi. "The Consequences of Mortgage Credit Expansion: Evidence from the U.S. Mortgage Default Crisis." *Quarterly Journal of Economics*, vol. 124, no. 4, Nov. 2009, pp. 1449–1496, academic.oup.com/qje/article-abstract/124/4/1449/1917185?redirectedFrom=fulltext, https://doi.org/10.1162/qjec.2009.124.4.1449.

[10]:
```
# Run the following to render to PDF
!jupyter nbconvert --to pdf project2.ipynb
```

```
[NbConvertApp] Converting notebook project2.ipynb to pdf
[NbConvertApp] Writing 443083 bytes to project2.pdf
```

m Created in Deepnote