ofinance-management-practices-code

October 10, 2024

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import RobustScaler
import math
import itertools
import logging
pd.options.display.float_format = '{:.2f}'.format
np.set_printoptions(suppress=True)
plot = False
companyNames = ['annapurna', 'chaitanya' ,'muthoot' , 'belstar' , 'satin' ,␣
↪'creditaccess' , 'asirvad' , 'spandanassphoorty' , 'fusionfinance'␣
↪,'bandhan']
finalResults = {}
projections = {}
totalBorrowersMicrofinance = 55000000

#Formula results below for companies with either upward or downwards sloping␣
↪cumulative borrower curves
positive_companies = ['annapurna', 'chaitanya', 'muthoot', 'satin']
negative_companies = ['creditaccess', 'spandanassphoorty']
no_trend_companies = ['belstar', 'asirvad' , 'fusionfinance','bandhan']
```

[2]:
```
# Load the Excel file
file_path = './Microfinance Data 21.xlsx'
excel_data = pd.ExcelFile(file_path)

df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0])


# Set Index of Data Frame to years
df.index = np.arange(1999, 2024)

projections = {}  # Dictionary to store all made projections
```

[3]:
```
# Prototype function that takes in two ARRAYS and a boolean (lower) as inputs␣
 ↪and fits the best polynomial regression model to it,
# Boolean (lower) is to determine whether to lower degree even further to avoid␣
 ↪overfitting
# Limits degrees to 3 to prevent overfitting

def best_polynomial_fit(X, y, lower):


    best_degree = 1
    best_r2 = -np.inf # Minus Infinity
    best_model = None

    # If statement to determine whether or not to lower degree

    if lower:
      degree = 2
    else:
      degree = 3

    # Run loop to determine how well each polynomial regression model works and␣
 ↪to utilise it

    for degree in range(1, degree+1):
        poly = PolynomialFeatures(degree)
        X_poly = poly.fit_transform(X)
        model = LinearRegression().fit(X_poly, y)
        r2 = r2_score(y, model.predict(X_poly)) # How well regression line␣
 ↪approximates data

        if r2 > best_r2:  # If approximates better switch best model to current␣
 ↪model
            best_r2 = r2
            best_degree = degree
            best_model = model




    return best_degree, best_model, best_r2
```

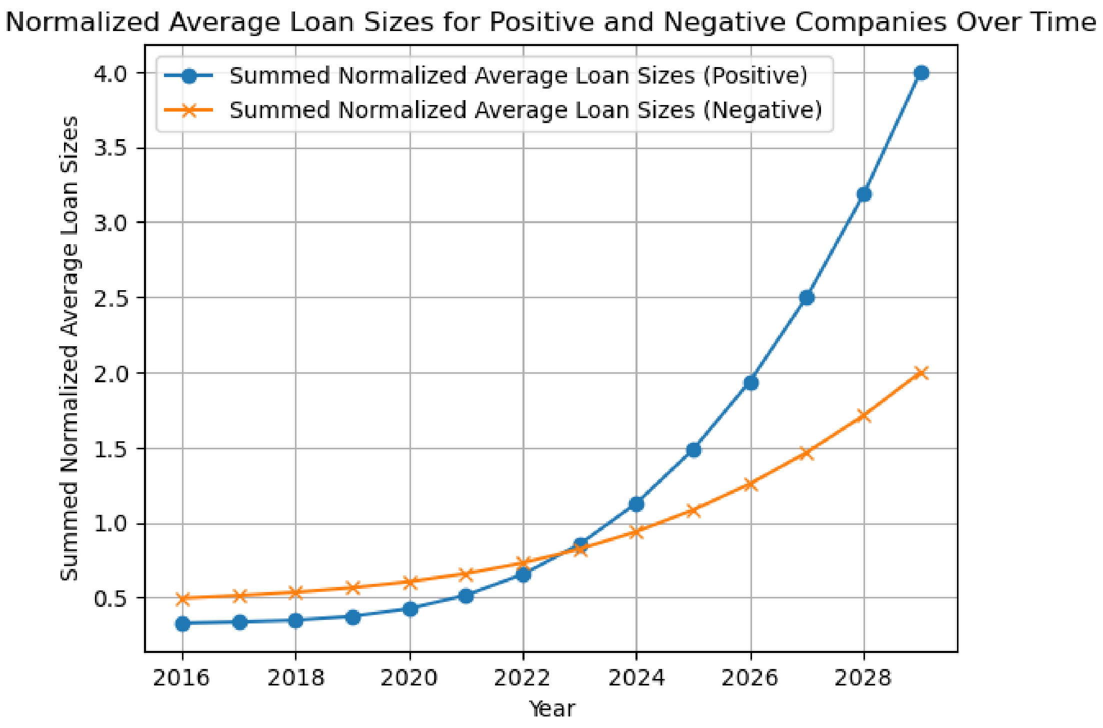
[4]:
```
def predict(dfKey, title, percent):

    y = df[dfKey].dropna().values.reshape(-1, 1)

    firstYear = df[dfKey].first_valid_index()
    lastYear = df[dfKey].last_valid_index()
    print(firstYear, lastYear)
    yearsTill2017 = 2017 - firstYear


    noNegative = True
    previousNum = 0
    numberOfValues = 0
    for num in df[dfKey]:

      if  math.isnan(num):
        continue
      numberOfValues += 1
      if num >= previousNum:

        previousNum = num
      else:
        noNegative = False

    yearsCurrent = np.arange(firstYear, lastYear+1).reshape(-1, 1)
    years_future_local =  np.arange(firstYear, 2031).reshape(-1, 1)



    best_degree, best_model, best_r2 = best_polynomial_fit(yearsCurrent, y,␣
↪False)

    poly = PolynomialFeatures(best_degree)
    future_X_poly = poly.fit_transform(years_future_local)
    projection = best_model.predict(future_X_poly)

    if (percent and projection[-1] >= 100) or␣
↪(df[dfKey][lastYear]>projection[-1] and noNegative):
        best_degree, best_model, best_r2 = best_polynomial_fit(yearsCurrent, y,␣
↪True)

        poly = PolynomialFeatures(best_degree)
        future_X_poly = poly.fit_transform(years_future_local)
        projection = best_model.predict(future_X_poly)


    if not ('Borrowers' in dfKey):
      projections[dfKey] = projection[yearsTill2017:]
    else:
      projections[dfKey] = projection[yearsTill2017-1:]
    if plot:
        plt.figure(figsize=(12, 6))
        plt.plot(years_future_local, projection, color='blue',␣
↪label=f'Projected {title} (Degree {best_degree})')
        plt.scatter(yearsCurrent, y, color='red', label='Actual Data')
        plt.title(title)
        plt.xlabel('Years')
        plt.grid(True)
        plt.legend()
        plt.show()
```

[5]:
```
for companyName in positive_companies:
    ␣
↪predict(f'{companyName}AverageLoanSize',f'{companyName}AverageLoanSize',False)
    ␣
↪predict(f'{companyName}ExpensePerBorrower',f'{companyName}ExpensePerBorrower',False)
    ␣
↪predict(f'{companyName}BorrowersPerBranch',f'{companyName}BorrowersPerBranch',False)
for companyName in negative_companies:
    ␣
↪predict(f'{companyName}AverageLoanSize',f'{companyName}AverageLoanSize',False)
    ␣
↪predict(f'{companyName}ExpensePerBorrower',f'{companyName}ExpensePerBorrower',False)
    ␣
↪predict(f'{companyName}BorrowersPerBranch',f'{companyName}BorrowersPerBranch',False)
```

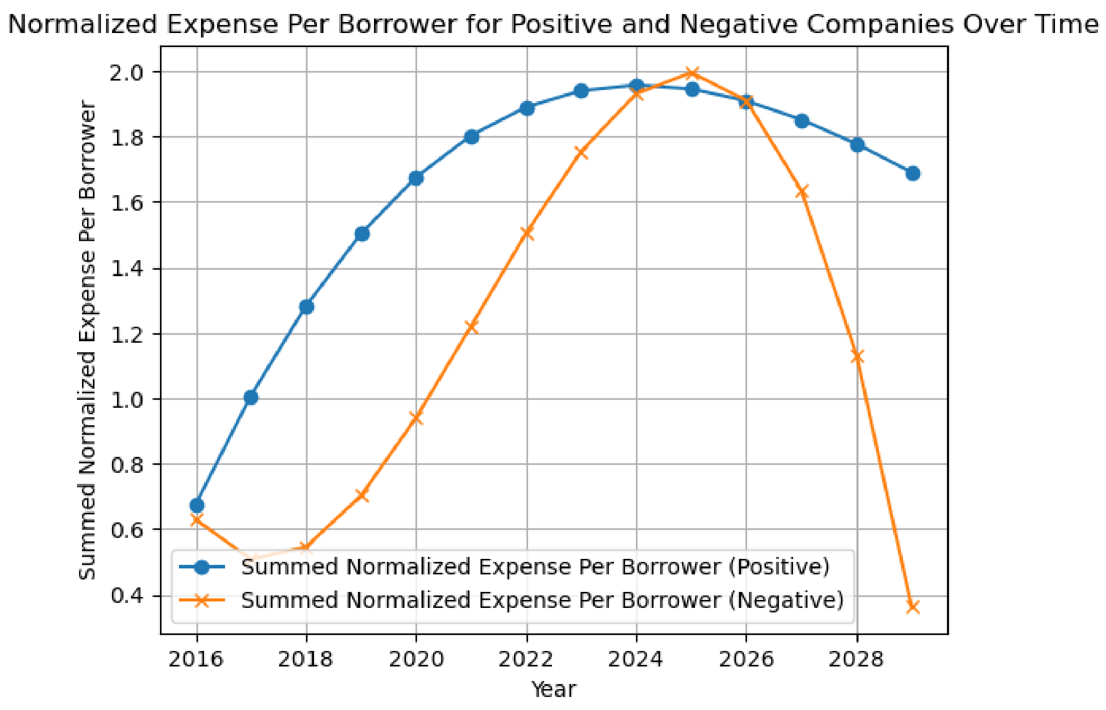
```
2014 2023
2016 2023
2015 2023
2016 2023
2016 2023
2011 2023
2016 2023
2016 2023
2016 2023
2016 2023
2016 2023
2015 2023
2014 2023
2017 2023
2012 2023
2016 2023
2016 2023
2014 2023
```

[6]:
```
years = np.arange(2016, 2030)

# Initialize arrays to hold the summed normalized loan sizes for positive and␣
↪negative companies
summed_normalized_loan_sizes_positive = np.zeros(len(years))
summed_normalized_loan_sizes_negative = np.zeros(len(years))

# Process each positive company
for companyName in positive_companies:
    # Access the saved AverageLoanSize data
    average_loan_sizes = np.array(projections[f'{companyName}AverageLoanSize']).
↪flatten()

    # Create a mask to filter out NaNs
    valid_mask = ~np.isnan(average_loan_sizes)

    # Normalize and accumulate the loan sizes for the current company
    if np.any(valid_mask):  # Check if there are any valid data points
        normalized_loan_sizes = average_loan_sizes[valid_mask] / np.
↪nanmax(average_loan_sizes[valid_mask])
        summed_normalized_loan_sizes_positive[valid_mask] +=␣
↪normalized_loan_sizes

# Process each negative company
for companyName in negative_companies:
    # Access the saved AverageLoanSize data
    average_loan_sizes = np.array(projections[f'{companyName}AverageLoanSize']).
↪flatten()

    # Create a mask to filter out NaNs
    valid_mask = ~np.isnan(average_loan_sizes)

    # Normalize and accumulate the loan sizes for the current company
    if np.any(valid_mask):  # Check if there are any valid data points
        normalized_loan_sizes = average_loan_sizes[valid_mask] / np.
↪nanmax(average_loan_sizes[valid_mask])
        summed_normalized_loan_sizes_negative[valid_mask] +=␣
↪normalized_loan_sizes

# Plot the summed normalized average loan sizes for all positive and negative␣
↪companies
plt.plot(years, summed_normalized_loan_sizes_positive, label='Summed Normalized␣
↪Average Loan Sizes (Positive)', marker='o')
plt.plot(years, summed_normalized_loan_sizes_negative, label='Summed Normalized␣
↪Average Loan Sizes (Negative)', marker='x')
```

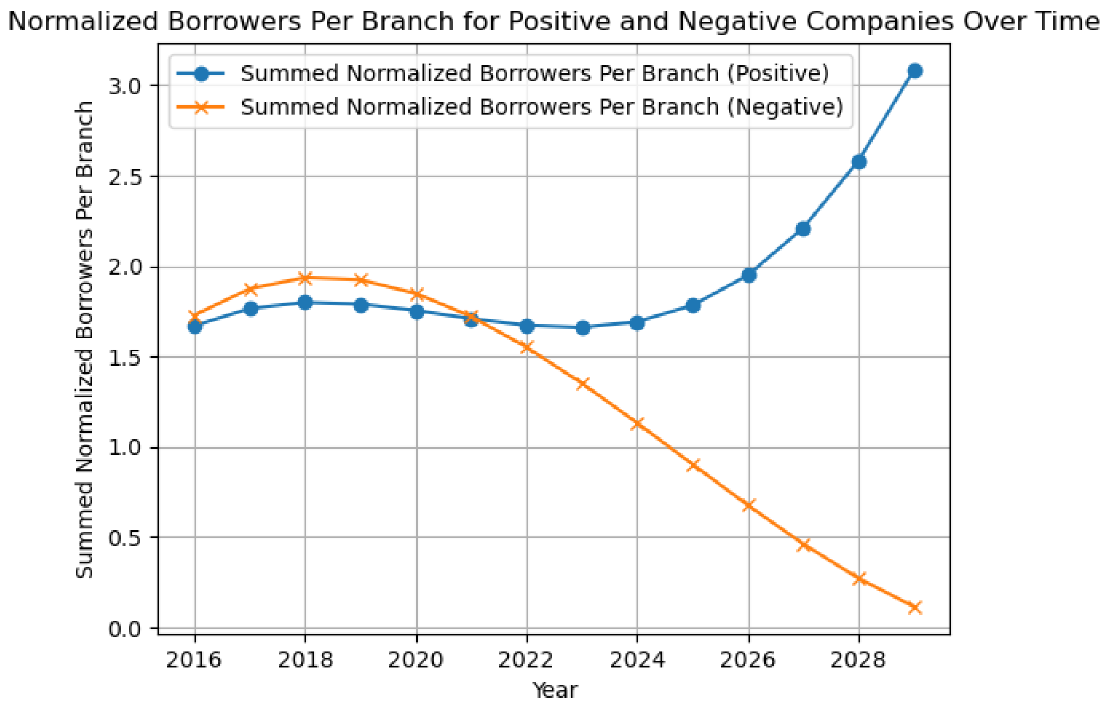
```
plt.xlabel('Year')
plt.ylabel('Summed Normalized Average Loan Sizes')
plt.title('Normalized Average Loan Sizes for Positive and Negative Companies␣
  ↪Over Time')
plt.legend()
plt.grid(True)
plt.show()
```



[7]:
```
years = np.arange(2016, 2030)  # Adjust based on the length of your data

# Initialize arrays to hold the summed normalized expense per borrower for␣
  ↪positive and negative companies
summed_normalized_expense_positive = np.zeros(len(years))
summed_normalized_expense_negative = np.zeros(len(years))

# Process each positive company for ExpensePerBorrower
for companyName in positive_companies:
    # Access the saved ExpensePerBorrower data
    expense_per_borrower = np.
  ↪array(projections[f'{companyName}ExpensePerBorrower']).flatten()

    # Create a mask to filter out NaNs
    valid_mask = ~np.isnan(expense_per_borrower)
    # Normalize and accumulate the expense for the current company
    if np.any(valid_mask):  # Check if there are any valid data points
        normalized_expense = expense_per_borrower[valid_mask] / np.
 ↪nanmax(expense_per_borrower[valid_mask])
        summed_normalized_expense_positive[valid_mask] += normalized_expense

# Process each negative company for ExpensePerBorrower
for companyName in negative_companies:
    # Access the saved ExpensePerBorrower data
    expense_per_borrower = np.
 ↪array(projections[f'{companyName}ExpensePerBorrower']).flatten()

    # Create a mask to filter out NaNs
    valid_mask = ~np.isnan(expense_per_borrower)

    # Normalize and accumulate the expense for the current company
    if np.any(valid_mask):  # Check if there are any valid data points
        normalized_expense = expense_per_borrower[valid_mask] / np.
 ↪nanmax(expense_per_borrower[valid_mask])
        summed_normalized_expense_negative[valid_mask] += normalized_expense

# Plot the summed normalized expense per borrower for all positive and negative␣
 ↪companies
plt.plot(years, summed_normalized_expense_positive, label='Summed Normalized␣
 ↪Expense Per Borrower (Positive)', marker='o')
plt.plot(years, summed_normalized_expense_negative, label='Summed Normalized␣
 ↪Expense Per Borrower (Negative)', marker='x')

plt.xlabel('Year')
plt.ylabel('Summed Normalized Expense Per Borrower')
plt.title('Normalized Expense Per Borrower for Positive and Negative Companies␣
 ↪Over Time')
plt.legend()
plt.grid(True)
plt.show()
```



[8]:
```python
# Assuming 'years' is an array or list containing the years starting from 2016
years = np.arange(2016, 2016 + len(next(iter(projections.values()))))  # Adjust␣
↪based on the length of your data

# Initialize arrays to hold the summed normalized borrowers per branch for␣
↪positive and negative companies
summed_normalized_borrowers_positive = np.zeros(len(years))
summed_normalized_borrowers_negative = np.zeros(len(years))

# Process each positive company for BorrowersPerBranch
for companyName in positive_companies:
    # Access the saved BorrowersPerBranch data
    borrowers_per_branch = np.
↪array(projections[f'{companyName}BorrowersPerBranch']).flatten()

    # Adjust the length of borrowers_per_branch to match 'years' if necessary
    if len(borrowers_per_branch) > len(years):
        borrowers_per_branch = borrowers_per_branch[:len(years)]
    elif len(borrowers_per_branch) < len(years):
        borrowers_per_branch = np.pad(borrowers_per_branch, (0, len(years) -␣
↪len(borrowers_per_branch)), constant_values=np.nan)

    # Create a mask to filter out NaNs
    valid_mask = ~np.isnan(borrowers_per_branch)
    # Normalize and accumulate the borrowers per branch for the current company
    if np.any(valid_mask):  # Check if there are any valid data points
        normalized_borrowers = borrowers_per_branch[valid_mask] / np.
 ↪nanmax(borrowers_per_branch[valid_mask])
        summed_normalized_borrowers_positive[valid_mask] += normalized_borrowers

# Process each negative company for BorrowersPerBranch
for companyName in negative_companies:
    # Access the saved BorrowersPerBranch data
    borrowers_per_branch = np.
 ↪array(projections[f'{companyName}BorrowersPerBranch']).flatten()

    # Adjust the length of borrowers_per_branch to match 'years' if necessary
    if len(borrowers_per_branch) > len(years):
        borrowers_per_branch = borrowers_per_branch[:len(years)]
    elif len(borrowers_per_branch) < len(years):
        borrowers_per_branch = np.pad(borrowers_per_branch, (0, len(years) -␣
 ↪len(borrowers_per_branch)), constant_values=np.nan)

    # Create a mask to filter out NaNs
    valid_mask = ~np.isnan(borrowers_per_branch)

    # Normalize and accumulate the borrowers per branch for the current company
    if np.any(valid_mask):  # Check if there are any valid data points
        normalized_borrowers = borrowers_per_branch[valid_mask] / np.
 ↪nanmax(borrowers_per_branch[valid_mask])
        summed_normalized_borrowers_negative[valid_mask] += normalized_borrowers

# Plot the summed normalized borrowers per branch for all positive and negative␣
 ↪companies
plt.plot(years, summed_normalized_borrowers_positive, label='Summed Normalized␣
 ↪Borrowers Per Branch (Positive)', marker='o')
plt.plot(years, summed_normalized_borrowers_negative, label='Summed Normalized␣
 ↪Borrowers Per Branch (Negative)', marker='x')

plt.xlabel('Year')
plt.ylabel('Summed Normalized Borrowers Per Branch')
plt.title('Normalized Borrowers Per Branch for Positive and Negative Companies␣
 ↪Over Time')
plt.legend()
plt.grid(True)
plt.show()
```


Normalized Borrowers Per Branch for Positive and Negative Companies Over Time
Summed Normalized Borrowers Per Branch (Positive)
Summed Normalized Borrowers Per Branch (Negative)
Summed Normalized Borrowers Per Branch
Year
0.0
0.5
1.0
1.5
2.0
2.5
3.0
2016
2018
2020
2022
2024
2026
2028