Klasyfikacja problemów ze spłatą kredytu – klasyfikacja w zbiorach niezbalansowanych

1.Wstęp

Analiza tzw. „fraudów" to jeden z głównych problemów klasyfikacyjnych – w jaki sposób skonstruować model i jakie metody zastosować, aby sklasyfikować poprawnie analizowane dane. W przypadku takiej analizy należy zwrócić uwagę na dwa główne czynniki:
- Czy niepoprawne sklasyfikowanie fraudu (w istocie wystąpił, a określiliśmy że nie nastąpił) czy informacji wiąże się z kosztami i jakie są to koszty.
- Czy niepoprawne sklasyfikowanie braku fraudu (fraud nie wystąpił, a określiliśmy że nastąpił) wiąże się z kosztami i jakie są to koszty.

W przypadku analizowanego przeze mnie przykładu, pierwszy problem jest zdecydowanie bardziej gorszy – w przypadku gdy osoba będzie miała problem ze spłatą kredytu, wiąże się to z kosztami dla banku, w przypadku drugim – jeżeli nie przyznamy osobie kredytu, twierdząc że może mieć ona problemy – narażamy się na to, że klient będzie niezadowolony, nie jest to jednak koszt finansowy jaki jest ponoszony.

Dane pochodzą z raportu: https://data.world/lpetrocelli/czech-financial-dataset-real-anonymized-transactions
Zostały one uprzednio oczyszczone oraz przygotowane na potrzeby innego projektu w innym oprogramowaniu.
Jako, że liczba danych odnośnie zaciągniętych kredytów wynosi około 700, natomiast liczba osób, które miały problem ze spłatą wynosiła około 10%, będzie trzeba zastosować metody, które pomogą poprawnie sklasyfikować dane.

2.Lista zmiennych w pliku .csv

1.wielkosckredytu–wielkość kredytu jako iloczyn platnosci i czasu trwania kredytu

2.dataurodzenia–data urodzenia zaciągającego kredyt (nie bierze udziału przy modelowaniu)

3.datazalozenia–data założenia konta (nie bierze udziału przy modelowaniu)

4.datapozyczki–data zaciągnięcia pożyczki w banku (nie bierze udziału przy modelowaniu)

5.czastrwaniakredytu–czas trwania kredytu wyrażony w miesiącach

6.platnosci-płatności miesięczne kredytu

7.plec–płeć osoby zaciągającej kredyt

8.typkarty–typ karty klienta

9.stan_konta_przydzielanie_kredytu–stan na koncie na moment przydzielania kredytu

10.A4–liczba mieszkańców w regionie klienta

11.A5–liczba gmin o liczbie mieszkańców <499 w regionie klienta

12.A6-liczba gmin o liczbie mieszkańców 500-1999 w regionie klienta

13.A7-liczba gmin o liczbie mieszkańców 2000-9999 w regionie klienta

14.A8–liczba gmin o liczbie mieszkańców >10000 w regionie klienta

15.A9–liczba miast w regionie klienta

16.A10–wskaźnik urbanizacji w regionie klienta

17.A11–średnia pensja w regionie klienta

18.A12–wskaźnik bezrobocia w 1995 roku w regionie klienta

19.A13-wskaźnik bezrobocia w 1996 roku w regionie klienta

20.A14–liczba przedsiębiorców na 1000 mieszkańców w regionie klienta

21.A15–liczba przestępstw w 1995roku w regionie klienta

22.A16–liczba przestępstw w regionie klienta w 1996 roku

23.wiek_pozyczka–wiek wyrażony w latach w jakim klient zaciąga pożyczkę

24. staz_pozyczka–liczba lat, jaka upłynęła od założenia konta do zaciągnięcia kredytu

25.czestotliwosc_platnosci–jakiego rodzaju raty spłaca klient

26.Czy_problemy–zmienna celowa binarna, określająca czy nastąpił problem podczas spłacania kredytu

27.iletransakcji–liczba transakcji przeprowadzona przez klienta

28.ilecredit–ile transakcji wpłat, powiększających stan konta zostało wykonanych przez klienta

29.ilewithdrawal–ile transakcji wypłat, zmniejszających stan konta zostało wykonanych przez klienta

30.ilecreditcardwithdrawal–ile transakcji wypłat za pomocą karty dla danego klienta

31.ilecreditincash–ile wpłat przy użyciu gotówki dla danego klienta

32.ilecollectionfromanotherbank–ile razy pieniądze przelane z innego banku dla danego klienta

33.ileremittancetoanotherbank–ile razy pieniądze zostały przelane do innego banku dla danego klienta

3. Wczytanie danych i określenie istotnych zmiennych, funkcje wykonujące klasyfikację

Wszystkie kody znajdują się w pliku .py.
Ważniejsze fragmenty opiszę poniżej:

```
import pandas as pd
import itertools
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.utils import resample
from sklearn.ensemble import RandomForestClassifier
from statistics import mean
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

danesas=pd.read_csv('danesasowe.csv',delimiter=';',decimal=',')
danesas=danesas.drop(['account_id','datazalozenia',
                'dataurodzenia','datazalozenia',
                'datapozyczki','status','disp_id'
                ,'typkarty','duration','payments','balance'],axis=1)

#Wybor kolumn ktore moga wejsc do modelu
kolumnyX=["platnosci","wielkosckredytu",
          "ileremittancetoanotherbank","ilewithdraw",
          "ilecollectionfromanotherbank","iletransakcji",
          "staz_pozyczka","ilecreditcardwithdrawal"]
X=danesas
y=X['Czy_problemy']
```

W pierwszej kolejności zostają wczytane moduły oraz do zmiennej „kolumnyX” przypisuję listę kolumn, które mogą wejść w skład modelu ostatecznego.

```
#Wszystkie mozliwe kombinacje kolumn - sprawdzenie ktore z kolumn daja najlepszy wynik
listapodzbiorowzmiennych=[]
for L in range(0, len(kolumnyX)+1):
    for subset in itertools.combinations(kolumnyX, L):
        listapodzbiorowzmiennych.append(list(subset))
```

W tym kroku tworzę wszystkie możliwe podzbiory kolumn zawartych w zmiennej „kolumnyX” posłuży to do analizy, które zestawy zmiennych osiągają dobre wartości końcowych parametrów. Ze względu na długość kompilacji oraz to, że poczatkowa ilość zmiennych w przypadku analizy jest zdecydowanie większa – wykorzystam jedynie zmienne, które były najbardziej skorelowane ze zmienną objaśnianą.

```
#Klasyfikacja
def Przeprowadz_Klasyfikacje(klasyfikatory,zmienna,X,y):
    Xt=X[zmienna]
    Xt=(Xt-Xt.mean())/Xt.std()
    #Xt=(Xt-Xt.min())/(Xt.max()-Xt.min())
    wartoscacctrain=[]
    wartoscacctest=[]
    wartoscroctrain=[]
    wartoscroctest=[]
    for key, classifier in klasyfikatory.items():
        for i in range(1):
            X_train, X_test, y_train, y_test = train_test_split(Xt, y, test_size=0.2,stratify=y.values)
            model = classifier.fit(X_train, y_train)
            ########################################
            step_factor = 0.05
            threshold_value = 0.2
            roc_score=0
            predicted_proba = model.predict_proba(X_train) #probability of prediction
            while threshold_value <=0.85: #continue to check best threshold upto probability 0.8
                temp_thresh = threshold_value
                predicted = (predicted_proba [:,1] >= temp_thresh).astype('int') #change the class boundary for prediction
                #print('Threshold',temp_thresh,'--',roc_auc_score(y_test, predicted))
                if roc_score<roc_auc_score(y_train, predicted): #store the threshold for best classification
                    roc_score = roc_auc_score(y_train, predicted)
                    thrsh_score = threshold_value
                threshold_value = threshold_value + step_factor
            #print('---Optimum Threshold ---',thrsh_score,'--ROC--',roc_score)
            ######################################
            y_pred_proba=model.predict_proba(X_train)
            y_pred = (y_pred_proba[:,1] >= thrsh_score).astype(bool)
```

Funkcja główna do przeprowadzenia klasyfikacji przyjmuje jako parametry kolejno:

-klasyfikatory – słownik, mający za klucz nazwę klasyfikatora, a za wartość – funkcję go wywołującą

-zmienna – lista zawierająca nazwy kolumn, które wchodzą w skład modelu

-X i y – wartości X i y które trafią do modelu. Nie będą to te same wartości za każdym razem, jako że odrębne metody będą wymagały pewnych transformacji X i y.

Dla każdego klasyfikatora przeprowadzamy klasyfikację, sprawdzając najpierw jaka wartość graniczna thresholda, daje najwyższą wartość ROC_AUC. Na tej podstawie dokonujemy predykcji. Analogicznie przeprowadzamy to dla zbioru testowego.

```
        training_score = cross_val_score(model, X_train, y_train, cv=5)
        test_score = cross_val_score(model, X_test, y_test, cv=5)
        wartoscacctrain.append(round(training_score.mean(), 2))
        wartoscacctest.append(round(test_score.mean(), 2))
        wartoscroctrain.append(roc_auc_score(y_train, y_pred))
        wartoscroctest.append(roc_auc_score(y_test, y_pred1))
    print(key,zmienna)
    print("Dane treningowe")
    print(classification_report(y_train, y_pred))
    print(confusion_matrix(y_train,y_pred))
    print()
    print("Dane testowe")
    print(classification_report(y_test, y_pred1))
    print(confusion_matrix(y_test,y_pred1))
    print("ROC TRAIN:",round(mean(wartoscroctrain),2),"ROC TEST:",round(mean(wartoscroctest),2))
    print("ACC TRAIN:",round(mean(wartoscacctrain),2),"ACC TEST:",round(mean(wartoscacctest),2))
    print()
```

Na koniec zostaną wyświetlone informacje o tym, jakiego klasyfikatora oraz jakiego zestawu zmiennych uzyliśmy, a także wyświetlone zostaną wartości jakości modelu, wartości ROC_AUC, trafność, a także macierz konfuzji i raport klasyfikacji, który pokaże wartości „recall” i „precision” które pomogą w określeniu błędów klasyfikacji wspomnianych we wstępie.

4. Klasyfikacja

```
#Zwykła klasyfikacja - zastosowanie jedynie wag#
import numpy as np
classifiers = {
    "LogisiticRegression": LogisticRegression(max_iter=30000,class_weight="balanced"),
    "DecisionTreeClassifier": DecisionTreeClassifier(class_weight="balanced"),
    "RandomForestClassifier": RandomForestClassifier(class_weight="balanced")
}
Przeprowadz_Klasyfikacje(classifiers,listapodzbiorowzmiennych[-1],X,y)
```

Pierwszym ze sposobów klasyfikacji będzie zwykła klasyfikacja która przyjmie parametr „balanced" do parametru „class_weight". Paramer ten przy wartości „balanced" określa jaka waga zostanie przypisana do klasy według wzoru:

```
n_samples / (n_classes * np.bincount(y))
```

Pomoże to w klasyfikacji zmiennych, które są w istocie fraudami kredytowymi, bez tego, prawie wszystkie zmienne zostaną przypisane jako takie, w których problem nie wystąpił. Spowoduje to, że trafność modelu będzie wynosić nawet 90%, jednak model będzie kompletnie nieprzydatny.

Omówię jeden z wyników wyświetlanych przez to wywołanie:

```
LogisiticRegression ['platnosci', 'wielkosckredytu', 'ileremittancetoanotherbank', 'ilewithdraw', 'ilecollectionfromanotherban
k', 'iletransakcji', 'staz_pozyczka', 'ilecreditcardwithdrawal']
Dane treningowe
              precision    recall  f1-score   support

           0       0.94      0.73      0.82       484
           1       0.23      0.66      0.34        61

    accuracy                           0.72       545
   macro avg       0.59      0.69      0.58       545
weighted avg       0.86      0.72      0.77       545

[[352 132]
 [ 21  40]]

Dane testowe
              precision    recall  f1-score   support

           0       0.93      0.89      0.91       122
           1       0.35      0.47      0.40        15

    accuracy                           0.85       137
   macro avg       0.64      0.68      0.66       137
weighted avg       0.87      0.85      0.86       137

[[109  13]
 [  8   7]]
ROC TRAIN: 0.69 ROC TEST: 0.68
ACC TRAIN: 0.66 ACC TEST: 0.64
```

Dla następującego modelu możemy zauważyć iż:

- wartości „precision" i „recall" zarówno w przypadku danych treningowych jak i testowych są dosyć wysokie, chociaż w przypadku „recall" – oczekiwalibyśmy większych wartości.
Okreslając „recall" jako zdolność modelu do wykrywania obserwacji danego rodzaju, natomiast „precision" jako zdolności modelu do wykrywania obserwacji w przypadku danego rodzaju poprawnie, widać że model wykrywa około 70% fraudów, natomiast klasyfikuje dużą ilość obserwacji jako fałszywie pozytywne – w rzeczywistości fraudu nie było, a model mimo wszystko go wykrył.

Widać to dobrze w przypadku danych treningowych:

-W 352 obserwacjach nie było fraudu i go nie wykryto
-W 40 przypadkach fraud był i go wykryto
-W 21 przypadkach fraud był i go nie wykryto
-W 132 przypadkach fraudu nie było, a go wykryto

W przypadku danych testowych, wygląda to gorzej:

```
[[109  13]
 [  8   7]]
```

Więcej niż połowa obserwacji gdzie problem z kredytem był, nie została wykryta.
Dla dwóch kolejnych modelu kolejno drzewa decyzyjnego i lasu losowego, dla danych treningowych dopasowanie jest perfekcyjne, natomiast dla danych testowych jest gorsze, niż w przypadku klasyfikacji logistycznej – pomijamy około 80% przypadków, gdzie wystąpił problem ze spłatą.

UPSAMPLING

W przypadku takich problemów, należy wykonywać pewne operacje na danych wejściowych. Jest kilka metod, które mogą pomóc w przypadku takiej sytuacji – można obniżyć ilość obserwacji, balansując do klasy, która ma niższą liczebność – takie działanie nazywa się undersamplingiem.
W przypadku modelu, który konstruuje jest to niestety bez sensu – ilość informacji która byłaby utracona porzuceniem znacznej ilości obserwacji tylko pogorszyłaby model. Należy więc „doprodukować" pewną ilość informacji odnośnie problemów kredytowych. Taką metodę nazywamy upsamplingiem, a sam upsampling możemy wykonać dwojako:

-dodając obserwacje, które już mamy
-dodając obserwacje, które sztucznie wygenerujemy

1. Upsampling z wykorzystaniem danych posiadanych

```
# UPSAMPLING #
import numpy as np
df_minority=X[X['Czy_problemy']==1]
df_majority=X[X['Czy_problemy']==0]
df_minority_upsampled=resample(df_minority,replace=True,n_samples=len(df_majority))
df_upsampled=pd.concat([df_minority_upsampled,df_majority])
y_upsampled=df_upsampled['Czy_problemy']
X_upsampled=df_upsampled.drop(['Czy_problemy'],axis=1)
classifiers = {
    "LogisiticRegression": LogisticRegression(max_iter=30000,class_weight="balanced"),
    "DecisionTreeClassifier": DecisionTreeClassifier(class_weight="balanced",max_depth=6,criterion='gini'),
    "RandomForestClassifier": RandomForestClassifier(class_weight="balanced")
}
Przeprowadz_Klasyfikacje(classifiers,listapodzbiorowzmiennych[-1],X_upsampled,y_upsampled)
```

W pierwszym przypadku, dodajemy taką ilość obserwacji z problemami kredytowymi, aby było ich tyle, co tych gdzie problemy nie wystąpiły.

Wyniki takiego działania dla drzewa decyzyjnego:

```
DecisionTreeClassifier ['platnosci', 'wielkosckredytu', 'ileremittancetoanotherbank', 'ilewithdraw', 'ilecollectionfromanotherb
ank', 'iletransakcji', 'staz_pozyczka', 'ilecreditcardwithdrawal']
Dane treningowe
              precision    recall  f1-score   support

           0       0.98      0.77      0.86       484
           1       0.81      0.98      0.89       485

    accuracy                           0.88       969
   macro avg       0.90      0.88      0.88       969
weighted avg       0.90      0.88      0.88       969

[[374 110]
 [  8 477]]

Dane testowe
              precision    recall  f1-score   support

           0       0.99      0.69      0.81       122
           1       0.76      0.99      0.86       121

    accuracy                           0.84       243
   macro avg       0.87      0.84      0.84       243
weighted avg       0.87      0.84      0.84       243

[[ 84  38]
 [  1 120]]
ROC TRAIN: 0.79 ROC TEST: 0.73
ACC TRAIN: 0.75 ACC TEST: 0.71
```

Jak widać, w przypadku takiego działania zdecydowanie poprawiła się jakoś wszystkich parametrów, zarówno trafności, wartości ROC_AUC, jak i analiza macierzy konfuzji potwierdza, że w przypadku danych treningowych popełniamy bardzo mało błędów, gdzie klasyfikujemy błędnie faktycznie występujący problem, natomiast popełniamy ok. 20% błędnych prognoz kiedy problemu w istocie nie było. W przypadku danych testowych wygląda to bardzo podobnie. W przypadku klasyfikacji logistycznej model nie poprawia aż tak bardzo sytuacji, w przypadku lasu losowego sytuacja jest prawie identyczna jak przy drzewie losowym. Dosyć dobre wyniki otrzymano również w przypadku zastosowania klasyfikatora SVC:

```
Support Vector Classifier ['platnosci', 'wielkosckredytu', 'ileremittar
herbank', 'iletransakcji', 'staz_pozyczka', 'ilecreditcardwithdrawal']
Dane treningowe
              precision    recall  f1-score   support

           0       0.84      0.74      0.79       485
           1       0.77      0.86      0.81       484

    accuracy                           0.80       969
   macro avg       0.80      0.80      0.80       969
weighted avg       0.80      0.80      0.80       969

[[358 127]
 [ 67 417]]
```

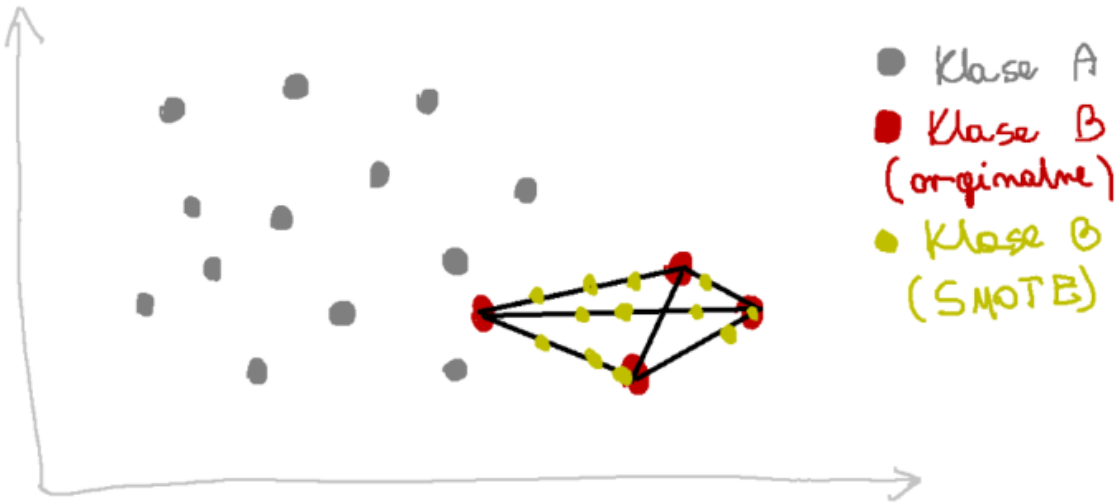
```
Dane testowe
              precision    recall  f1-score   support

           0       0.87      0.74      0.80       121
           1       0.77      0.89      0.83       122

    accuracy                           0.81       243
   macro avg       0.82      0.81      0.81       243
weighted avg       0.82      0.81      0.81       243

[[ 89  32]
 [ 13 109]]
AUC TRAIN: 0.79 AUC TEST: 0.78
ACC TRAIN: 0.76 ACC TEST: 0.73
```

W przypadku tej metody, otrzymane wyniki były dobre, jednak nie aż tak jak w przypadku drzew losowych. Jest to spowodowane tym, że w przypadku analizy z dodaniem sztucznie obserwacji, drzewa mają dużą tendencję do przeuczenia, jako że ich metoda podziału opiera się na nagromadzeniu informacji odnośnie pewnych zmiennych i ich zróżnicowania.

2. Smote upsampling

W przypadku zwykłego upsamplingu narażamy się na problem przeuczenia – model i jego ocena opiera się na tych samych informacjach, więc wychodząc poza obserwacje zawarte, szczególnie w przypadku tak małego zbioru informacji jak w tym przypadku.

Z pomocą przychodzi metoda SMOTE, czyli Synthetic Minority Oversampling Technique. Działanie metody bardzo dobrze obrazuje ten szkic:



Działanie metody:

-Wybieramy losowo przykładową obserwację dla klasy mniejszościowej.

-Znajdujemy k najbliższych sąsiadów dla wybranej obserwacji (z punktu 1).

-Wybieramy jednego z tych sąsiadów i umieszczamy syntetyczny punkt w dowolnym miejscu na linii łączącej rozpatrywany punkt z sąsiadem.

-Powtarzamy kroki 1-3, aż dane zostaną zbalansowane.

Zastosowanie metody

```
#SMOTE#
oversample = SMOTE(sampling_strategy=1,k_neighbors=5)
Xn, yn = oversample.fit_resample(X, y)
classifiers = {
    "LogisiticRegression": LogisticRegression(max_iter=30000,class_weight="balanced"),
    "DecisionTreeClassifier": DecisionTreeClassifier(class_weight="balanced",max_depth=6,criterion='gini'),
    "RandomForestClassifier": RandomForestClassifier(class_weight="balanced")
}
Przeprowadz_Klasyfikacje(classifiers,listapodzbiorowzmiennych[-1],Xn,yn)
```

Ponownie przedstawię przykładowy wynik wywołania funkcji:

```
DecisionTreeClassifier ['platnosci', 'wielkosckredytu', 'ileremittancetoanotherbank', 'ilewithdraw', 'ilecollectionfromanotherb
ank', 'iletransakcji', 'staz_pozyczka', 'ilecreditcardwithdrawal']
Dane treningowe
              precision    recall  f1-score   support

           0       0.90      0.79      0.84       484
           1       0.81      0.91      0.86       485

    accuracy                           0.85       969
   macro avg       0.85      0.85      0.85       969
weighted avg       0.85      0.85      0.85       969

[[382 102]
 [ 44 441]]

Dane testowe
              precision    recall  f1-score   support

           0       0.84      0.67      0.75       122
           1       0.72      0.87      0.79       121

    accuracy                           0.77       243
   macro avg       0.78      0.77      0.77       243
weighted avg       0.78      0.77      0.77       243

[[ 82  40]
 [ 16 105]]
ROC TRAIN: 0.79 ROC TEST: 0.75
ACC TRAIN: 0.74 ACC TEST: 0.7
```

Widać, że osiągnięty wynik jest bardzo podobny w przypadku zwykłego upsamplingu, jednak tutaj jest zdecydowanie mniejsza szansa na to, że model się przeuczył, to zastosowanie poprawia również oszacowanie w przypadku klasyfikacji logistycznej.

W przypadku ostatniej metody, jako że same dodawanie zmiennych sztucznych jest oparte na algorytmie najbliższych sąsiadów, przeprowadziłem również analizę dla modelu k-najbliższych sąsiadów.

```
Dane testowe
              precision    recall  f1-score   support

           0       0.91      0.73      0.81       122
           1       0.77      0.93      0.84       121

    accuracy                           0.83       243
   macro avg       0.84      0.83      0.83       243
weighted avg       0.84      0.83      0.83       243

[[ 89  33]
 [  9 112]]
AUC TRAIN: 0.86 AUC TEST: 0.79
ACC TRAIN: 0.78 ACC TEST: 0.71
```

Istotnie, uzyskane z pomocą tego modelu oszacowanie jest dobre, większość obserwacji z problemami ze spłatą została wykryta.

5. Klasyfikacja z pomocą syntetycznego klasyfikatora

Dodatkowo, jednym z rozwiązań jakie można zastosować w przypadku klasyfikacji jest utworzenie sztucznego klasyfikatora, jako pewnej wypadkowej klasyfikatorów, wykorzystując prawdopodobieństwa, że dana obserwacja będzie danej klasy.

```python
#Wlasny klasyfikator

def Wlasny_klasyfikator(klasyfikatory,zmienna,X,y):
    Xt=X[zmienna]
    Xt=(Xt-Xt.mean())/Xt.std()
    #Xt=(Xt-Xt.min())/(Xt.max()-Xt.min())
    wartoscacctrain=[]
    wartoscacctest=[]
    wartoscroctrain=[]
    wartoscroctest=[]
    predictprobatrain=[]
    predictprobatest=[]
    for key, classifier in klasyfikatory.items():
        X_train, X_test, y_train, y_test = train_test_split(Xt, y, test_size=0.2,stratify=y.values)
        model = classifier.fit(X_train, y_train)
        y_pred_proba=model.predict_proba(X_train)
        predictprobatrain.append(y_pred_proba)

        y_pred_probatest=model.predict_proba(X_test)
        predictprobatest.append(y_pred_probatest)

    new_proba_train=sum(predictprobatrain,0)/len(klasyfikatory)
    new_proba_train=sum(predictprobatest,0)/len(klasyfikatory)
    y_pred = (y_pred_proba[:,1] >= 0.5).astype(bool)
    y_pred1=(model.predict_proba(X_test)[:,1]>=0.5).astype(bool)

    print("Dane treningowe")
    print(classification_report(y_train, y_pred))
    print(confusion_matrix(y_train,y_pred))
    print()
    print("Dane testowe")
    print(classification_report(y_test, y_pred1))
    print(confusion_matrix(y_test,y_pred1))
```

W tym celu, utworzona została odrębna funkcja, która z pomocą wielu klasyfikatorów tworzy wypadkową prawdopodobieństwa i na tej podstawie dokonuje klasyfikacji. W funkcji obrano threshold = 0.5, natomiast jest to również możliwe, aby manipulować tym, w zależności kierunku siły oszacowania jaki chcemy otrzymać.

Wywołanie funkcji:

```
oversample = SMOTE(sampling_strategy=1,k_neighbors=5)
Xn, yn = oversample.fit_resample(X, y)
Xn=Xn
yn=yn
classifiers = {
    "LogisiticRegression": LogisticRegression(max_iter=30000,class_weight="balanced"),
    "KNearest": KNeighborsClassifier(n_neighbors=5),
    "Support Vector Classifier": SVC(kernel='rbf',probability=True,class_weight="balanced")
}
Przeprowadz_Klasyfikacje1(classifiers,listapodzbiorowzmiennych[-1],Xn,yn)
```

Otrzymane wyniki:

Dane treningowe

	precision	recall	f1-score	support
0	0.84	0.74	0.79	484
1	0.77	0.86	0.81	485
accuracy			0.80	969
macro avg	0.81	0.80	0.80	969
weighted avg	0.81	0.80	0.80	969

```
[[358 126]
 [ 66 419]]
```

Dane testowe

	precision	recall	f1-score	support
0	0.90	0.73	0.81	122
1	0.77	0.92	0.84	121
accuracy			0.82	243
macro avg	0.83	0.82	0.82	243
weighted avg	0.84	0.82	0.82	243

```
[[ 89  33]
 [ 10 111]]
```

Precyzja takiego oszacowania jest dosyć dobra, widać, że w przypadku obserwacji gdzie problem ze spłatą wystąpił, to większość takich obserwacji została wykryta.

6. Możliwe zmiany i poprawki do poprawy modelu

Oczywistym jest, że analiza problemów kredytowych w przypadku tak małej ilości informacji jest bardzo trudna, jeżeli nie niemożliwa. Należałoby zebrać znacznie większą liczbę informacji odnośnie obu grup. Ponadto, analiza została przeprowadzona na niezmienonych parametrach klasyfikatorów – kilka z nich zostało przetestowanych poza zaprezentowanymi tutaj zrzutami ekranu, natomiast nie zostały one sprawdzone na tyle dogłębnie aby wyciągnąć wnioski. Stąd, kolejne kroki które możnaby było przeprowadzić:
- Zebranie większej ilości informacji
- Zmiany hiperparametrów w przypadku klasyfikacji logistycznej
- Zmiany sposobów podziału drzew, ilości liści, kryterium dzielenia w przypadku klasyfikacji drzewami i za pomocą lasu losowego
- Dokładna analiza **wszystkich** kombinacji zmiennych – ze względu na długość kompilacji i sposób wywoływania krok pominąłem: ilość modeli które należało by sprawdzić wyniesie 2^x-1, gdzie x – liczba wszystkich zmiennych.

-Wielokrotna analiza z pomocą tych samych modeli i dopisywanie wartości prawdopodobieństwa przy użyciu klas sztucznych, do klas prawdziwych – jest to jeden z pomysłów do których zastosowania potrzeba by zdecydowanie więcej czasu. Do danych, które byłyby sztucznie generowane z pomocą metody SMOTE należałoby dodać pewny „label" który pomagałby określić, czy obserwacja pochodzi z sztucznego dodania, czy jest obserwacją naturalną. Wtedy, dla wielokrotnej klasyfikacji (gdzie w każdym kolejnym wywołaniu, punkty sztuczne zmienią się) przypisywać do konkretnych obserwacji szansę, że w tym kroku byłyby one przypisane do klasy pierwszej. Wtedy, dla pewnego określonego współczynnika granicznego, następowałaby klasyfikacja.

Mimo wszystko, sposób w jaki jest przeprowadzona klasyfikacja, a także wyniki są pomocne przy analizie zjawiska określania problemów kredytowych, nawet pomimo tak małej liczby obserwacji.