MED – Metody Eksploracji Danych

"Segmentacja klientów e-commerce na podstawie danych o użytkownikach i ich zachowań zakupowych"

Bartosz Zaborowski 319996

1. Opis projektu

Celem projektu jest zastosowanie wybranych metod eksploracji danych do segmentacji klientów sklepu internetowego oraz oceny możliwości automatycznego przypisywania klientów do wyznaczonych segmentów.

W ramach projektu wykorzystano klasyczną analizę RFM (Recency, Frequency, Monetary) do opisu zachowań zakupowych klientów. Na podstawie tych cech dokonano segmentacji klientów z użyciem algorytmu k-means, a następnie przeprowadzono klasyfikację do segmentów za pomocą algorytmów:

- Drzewo decyzyjne (Decision Tree)
- Naiwny klasyfikator Bayesa (Naive Bayes)

2. Zbiór danych

W projekcie wykorzystano publiczny zbiór danych **Online Retail Dataset**, zawierający dane transakcyjne sklepu internetowego. Dane pochodzą ze strony Kaggle.

Każdy rekord w zbiorze reprezentuje pojedynczą pozycję na fakturze i zawiera m.in.:

- `InvoiceNo` – numer faktury
- `StockCode` – kod produktu
- `Description` – opis produktu
- `Quantity` – liczba zakupionych sztuk
- `InvoiceDate` – data zakupu
- `UnitPrice` – cena jednostkowa
- `CustomerID` – identyfikator klienta
- `Country` – kraj klienta

Zbiór danych zawiera około 500 000 rekordów transakcyjnych. Po wstępnym przetworzeniu i agregacji po `CustomerID` uzyskano zbiór danych na poziomie klientów (analiza RFM), liczący około 1300 unikalnych klientów.

3. Opis algorytmów

3.1 Analiza RFM

Analiza RFM jest metodą segmentacji klientów opartą na trzech wskaźnikach:

- **Recency (R)** – liczba dni od ostatniego zakupu klienta
- **Frequency (F)** – liczba transakcji (unikalnych faktur)
- **Monetary (M)** – łączna wartość zakupów klienta

Cechy RFM stanowią wejście do dalszych etapów analizy.

3.2 k-means (etap pomocniczy)

Algorytm k-means został użyty do wygenerowania segmentów klientów na podstawie cech RFM. Liczba klastrów została ustalona na 3. Uzyskane etykiety segmentów stanowią klasy docelowe dla algorytmów klasyfikacyjnych i na tym etapie przypisane zostały im wstępne robocze nazwy 1, 2, 3.

3.3 Drzewo decyzyjne

Drzewo decyzyjne to algorytm nadzorowanego uczenia maszynowego, który buduje model w postaci hierarchii reguł decyzyjnych. Algorytm ten jest łatwy do interpretacji i dobrze radzi sobie z danymi numerycznymi.

3.4 Naiwny klasyfikator Bayesa

Naiwny Bayes jest probabilistycznym klasyfikatorem opartym na twierdzeniu Bayesa i założeniu niezależności cech. W projekcie wykorzystano wariant Gaussian Naive Bayes, dostosowany do danych ciągłych.

4. Wybrane technologie

Projekt został zrealizowany w następującym środowisku technologicznym:

- **Python 3.12** – język programowania

- **pandas** – przetwarzanie i agregacja danych
- **numpy** – obliczenia numeryczne
- **scikit-learn** – implementacja algorytmów k-means, Decision Tree oraz Naive Bayes
- **matplotlib / seaborn** – wizualizacja wyników (macierze konfuzji)

5. Opis wstępnej implementacji

Poniżej przedstawię projekt implementacji rozwiązania wraz z opisem struktury projektu, kluczowych plików oraz najważniejszych fragmentów kodu. Implementacja ma charakter modularny i odzwierciedla pełny pipeline eksploracji danych – od wczytania danych transakcyjnych, przez ich przetwarzanie i segmentację klientów, aż po klasyfikację i ewaluację wyników.

5.1 Struktura projektu

Projekt posiada następującą strukturę katalogów:

```
MED/Projekt/
├── data/
│   └── OnlineRetail.csv
├── src/
│   ├── data_loader.py
│   ├── evaluation.py
│   ├── feature_engineering.py
│   ├── main.py
│   ├── preprocessing.py
│   ├── segmentation.py
│   └── classifiers/
│       ├── decision_tree.py
│       └── naive_bayes.py
```

Taki podział umożliwia czytelne rozdzielenie odpowiedzialności pomiędzy poszczególne etapy przetwarzania danych oraz łatwą rozbudowę projektu w przyszłości.

5.2 Wczytywanie danych – `data_loader.py`

Moduł `data_loader.py` odpowiada za wczytanie danych transakcyjnych z pliku CSV. Zastosowano odpowiednie kodowanie znaków, zgodne ze specyfiką zbioru Online Retail.

```
import pandas as pd

def load_data(path: str) -> pd.DataFrame:
    df = pd.read_csv(path, encoding="ISO-8859-1")
    return df
```

Moduł ten stanowi pierwszy krok pipeline'u i dostarcza dane wejściowe do dalszego przetwarzania.

5.3 Wstępne przetwarzanie danych – `preprocessing.py`

Celem preprocessing'u jest oczywiście oczyszczenie danych oraz przygotowanie ich do dalszej analizy. W module usuwane są:

- rekordy bez przypisanego identyfikatora klienta,
- transakcje o ujemnej liczbie sztuk (zwroty).

Dodatkowo kolumna `InvoiceDate` jest konwertowana do typu daty.

```
import pandas as pd

def preprocess(df: pd.DataFrame) -> pd.DataFrame:
    df = df.dropna(subset=["CustomerID"])
    df = df[df["Quantity"] > 0]
    df["InvoiceDate"] = pd.to_datetime(df["InvoiceDate"])
    return df
```

5.4 Inżynieria cech – analiza RFM (`feature_engineering.py`)

Moduł `feature_engineering.py` odpowiada za agregację danych transakcyjnych do poziomu klienta oraz obliczenie cech RFM (Recency, Frequency, Monetary).

```
import pandas as pd
import datetime as dt

def calculate_rfm(df: pd.DataFrame) -> pd.DataFrame:
    NOW = dt.datetime(2011, 12, 10)

    rfm = df.groupby("CustomerID").agg({
        "InvoiceDate": lambda x: (NOW - x.max()).days,
        "InvoiceNo": "nunique",
        "Quantity": "sum",
        "UnitPrice": "mean"
    })

    rfm.columns = ["Recency", "Frequency", "Quantity", "UnitPrice"]
    rfm["Monetary"] = rfm["Quantity"] * rfm["UnitPrice"]

    return rfm[["Recency", "Frequency", "Monetary"]]
```

Efektem działania modułu jest macierz cech, w której każdy wiersz odpowiada jednemu klientowi.

5.5 Segmentacja klientów – `segmentation.py`

Segmentacja klientów realizowana jest z użyciem algorytmu k-means. Przed klasteryzacją dane RFM są standaryzowane, aby zapewnić porównywalną skalę cech.

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import pandas as pd

def segment_customers(rfm: pd.DataFrame, n_clusters: int = 3):
    scaler = StandardScaler()
    rfm_scaled = scaler.fit_transform(rfm)

    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    segments = kmeans.fit_predict(rfm_scaled)

    rfm["Segment"] = segments
    return rfm, rfm_scaled
```

Wynikiem działania funkcji są:

- zbiór klientów z przypisanym numerem segmentu,
- znormalizowana macierz cech wykorzystywana w klasyfikacji.

5.6 Klasyfikatory – `decision_tree.py` i `naive_bayes.py`

W projekcie zastosowano dwa klasyczne algorytmy uczenia nadzorowanego: drzewo decyzyjne oraz naiwny klasyfikator Bayesa. Oba modele zostały zaimplementowane w postaci osobnych klas, co zapewnia czytelność kodu, spójny interfejs oraz możliwość łatwej rozbudowy lub wymiany algorytmów w przyszłości.

Klasyfikatory uczone są na danych wejściowych w postaci cech RFM (Recency, Frequency, Monetary), natomiast zmienną docelową są etykiety segmentów klientów wygenerowane wcześniej przez algorytm k-means. W ten sposób problem klasyfikacji polega na nauczeniu modeli odtwarzania segmentacji klientów na podstawie ich zachowań zakupowych.

Drzewo decyzyjne wykorzystuje mechanizm hierarchicznych reguł decyzyjnych, które dzielą przestrzeń cech na coraz bardziej jednorodne obszary. Model ten dobrze radzi sobie z danymi tablicowymi, nie wymaga skalowania cech oraz umożliwia nieliniowe rozdzielenie klas. W projekcie zastosowano domyślną konfigurację drzewa decyzyjnego z ustalonym ziarnem losowości w celu zapewnienia powtarzalności wyników.

Naiwny klasyfikator Bayesa opiera się na probabilistycznym podejściu i założeniu niezależności cech. W implementacji wykorzystano wariant Gaussa, odpowiedni dla cech ciągłych, takich jak wartości RFM. Model ten charakteryzuje się bardzo szybkim czasem uczenia oraz prostotą obliczeniową, jednak jego skuteczność może być ograniczona w przypadku silnej zależności pomiędzy cechami.

Oba klasyfikatory udostępniają jednolity interfejs obejmujący proces trenowania modelu oraz generowania predykcji, co umożliwia ich bezpośrednie porównanie na tym samym zbiorze testowym. Takie podejście pozwala ocenić, w jakim stopniu różne algorytmy są w stanie odtworzyć segmentację klientów uzyskaną metodą nienadzorowaną.

5.7 Ewaluacja modeli – `evaluation.py`

Moduł `evaluation.py` odpowiada za ocenę jakości klasyfikacji. Generowany jest raport klasyfikacji oraz macierz konfuzji w formie wizualnej.

```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

def evaluate_model(y_true, y_pred, title: str):
    print(title)
    print(classification_report(y_true, y_pred))

    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt="d")
    plt.title(title)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

5.8 Plik główny – `main.py`

Plik `main.py` integruje wszystkie opisane wcześniej moduły i realizuje pełny przebieg eksperymentu czyli:

```
# 1. Load data
# 2. Preprocess
# 3. Feature engineering (RFM)
# 4. Segmentation (k-means)
# 5. Train-test split
# 6. Classification (Decision Tree, Naive Bayes)
# 7. Evaluation
```

Stanowi on punkt wejścia do projektu oraz umożliwia łatwe uruchomienie całego procesu eksploracji danych i klasyfikacji klientów.

6. Opis wstępnych wyników

Dla zbioru testowego uzyskano następujące wyniki:

Drzewo decyzyjne

- Dokładność (accuracy): ~100%
- Bardzo dobra separacja segmentów
- Brak istotnych błędów klasyfikacji

Naiwny klasyfikator Bayesa

- Dokładność (accuracy): ~93%
- Dobre wyniki dla głównych segmentów
- Gorsza skuteczność dla najmniej licznej klasy 2

Macierze konfuzji potwierdzają wysoką skuteczność klasyfikatora drzewa decyzyjnego oraz poprawne, choć mniej precyzyjne, działanie Naive Bayes.

7. Wnioski

Przeprowadzone eksperymenty pokazują, że:

- Analiza RFM skutecznie opisuje zachowania klientów e-commerce
- Segmenty wygenerowane metodą k-means mogą być z powodzeniem wykorzystywane jako klasy docelowe
- Drzewo decyzyjne bardzo dobrze radzi sobie z klasyfikacją klientów na podstawie cech RFM
- Naiwny klasyfikator Bayesa jest prosty w implementacji, lecz mniej skuteczny dla niezbalansowanych danych