https://www.kaggle.com/code/anubhavgoyal10/airline-passenger-satisfaction-knn/notebook

Czemu ta baza danych -> dużo kolumn, dobre dane - mało kategorialnych, więcej numerycznych

Opis kodu:

1. Importowanie bibliotek:

- numpy: biblioteka do obliczeń numerycznych.
- pandas: biblioteka do manipulacji i analizy danych, prawdopodobnie używana do wczytania danych z plików CSV.
- matplotlib.pyplot i seaborn: biblioteki do tworzenia wizualizacji danych.
- Biblioteki scikit-learn:
    - SimpleImputer: klasa do obsługi brakujących wartości w danych.
    - OneHotEncoder i MinMaxScaler: klasy do przygotowania danych dla modeli uczenia maszynowego (kodowanie danych kategorycznych i skalowanie danych numerycznych).
    - Funkcje train_test_split i cross_val_score: służą do podziału danych na zbiór treningowy i testowy oraz oceny modeli.
    - GridSearchCV: klasa do optymalizacji hiperparametrów modeli.
    - Różne modele klasyfikacji maszynowej:
        - LogisticRegression, Perceptron, SGDClassifier: modele klasyfikacji liniowej.
        - SVC, LinearSVC: modele klasyfikacji Support Vector Machine (SVM).
        - RandomForestClassifier: model klasyfikacji oparty na zespole drzew decyzyjnych.
        - KNeighborsClassifier: model klasyfikacji K najbliższych sąsiadów.
        - GaussianNB: model klasyfikacji Naive Bayes.
        - DecisionTreeClassifier: model klasyfikacji oparty na drzewie decyzyjnym.
- google.colab (opcjonalnie): kod ten może być przeznaczony do uruchomienia w środowisku Google Colab, a ta linia montuje Twój dysk Google Drive.

2. Wczytywanie danych:

- Kod wykorzystuje pandas.read_csv do wczytania dwóch plików CSV z dysku Google Drive.
    - Jeden plik to dane treningowe, używane do budowy modelu.
    - Drugi plik to dane testowe, używane do oceny modelu.

3. Przetwarzanie wstępne danych:

- Kod zawiera kroki przetwarzania wstępnego danych, takie jak:
    - Uzupełnianie lub usuwanie brakujących wartości za pomocą SimpleImputer.
    - Kodowanie danych kategorycznych za pomocą OneHotEncoder.
    - Skalowanie danych numerycznych za pomocą MinMaxScaler.

4. Trening i ocena modeli:

- Kod dzieli dane treningowe na zbiór treningowy i walidacyjny przy użyciu train_test_split.
- Następnie różne modele klasyfikacji maszynowej są trenowane na zbiorze treningowym.
- Wyniki modeli są oceniane za pomocą funkcji cross_val_score lub podobnych.

- Kod może wykorzystywać GridSearchCV do znalezienia najlepszych hiperparametrów dla każdego modelu.

Analiza eksploracyjna danych z wykorzystaniem biblioteki pandas i seaborn

Ten fragment kodu wykorzystuje biblioteki pandas i seaborn do przeprowadzenia analizy eksploracyjnej danych zapisanych w plikach CSV. Kod pozwala na wstępną analizę danych z pliku "train.csv". Wygenerowane statystyki i wykresy pomagają zrozumieć rozkład i zależności między zmiennymi, co może być przydatne w dalszej analizie i budowaniu modeli uczenia maszynowego.

1. Wczytanie danych:

- df_train = pd.read_csv('/content/drive/Shareddrives/Wprowadzenie do AI - projekt/train.csv'):Wczytuje dane z pliku "train.csv" do zmiennej df_train.
- df_test = pd.read_csv('/content/drive/Shareddrives/Wprowadzenie do AI - projekt/test.csv'):Wczytuje dane z pliku "test.csv" do zmiennej df_test.

2. Eksploracja wstępna danych z df_train:

- df_train.head(): Wyświetla pierwsze kilka wierszy danych w df_train, co pozwala na szybkie sprawdzenie zawartości pliku.
- df_train.describe(): Generuje statystyki podsumowujące dla danych numerycznych w df_train, takie jak średnia, odchylenie standardowe, minimum i maksimum.
- df_train.info(): Pokazuje typy danych każdej kolumny w df_train oraz inne informacje o strukturze danych.
- df_train.describe().T: Transponuje wynik funkcji describe(), dzięki czemu uzyskuje się tabelę z nazwami kolumn jako wierszami i statystykami jako kolumnami.

3. Identyfikacja kolumn:

- cat_columns = df_train.select_dtypes(object).columns.to_list(): Tworzy listę zawierającą nazwy kolumn, które zawierają dane kategoryczne (typu object).
- num_columns = df_train.select_dtypes(np.number).columns.to_list(): Tworzy listę zawierającą nazwy kolumn, które zawierają dane numeryczne.

4. Wizualizacja rozkładu danych:

- plt.figure(figsize = (4, 5)): Tworzy nową figurę dla wykresów o określonej wielkości (4 na 5 cali).
- Pętla for col in num_columns iteruje po wszystkich kolumnach numerycznych:
    - sns.countplot(data=df_train, x=col, hue='satisfaction'): Tworzy wykres rozkładu dla każdej kolumny numerycznej, uwzględniając poziom zadowolenia ("satisfaction") jako kolor rozróżniający.

5. Wizualizacja rozkładu danych według poziomów zadowolenia:

- g = sns.FacetGrid(df_train, col='satisfaction'): Tworzy siatkę wykresów, gdzie kolumny odpowiadają poziomowi zadowolenia.

- g.map(plt.hist, 'Age', bins=10): Na każdej kolumnie siatki tworzy histogram wieku ("Age") z 10 przedziałami. Podobne operacje wykonywane są dla kolejnych kolumn: "Flight Distance", "Customer Type","Class".
- sns.countplot(df_train,x='Gender',hue='satisfaction'): Tworzy wykres rozkładu dla płci ("Gender"),uwzględniając poziom zadowolenia ("satisfaction") jako kolor rozróżniający.
- sns.countplot(df_train, x='Type of Travel', hue='satisfaction'): Tworzy wykres rozkładu dla typu podróży ("Type of Travel"), uwzględniając poziom zadowolenia ("satisfaction") jako kolor rozróżniający.

Kodowanie kolumny “satisfaction”

Kodując kolumnę "satisfaction", dane kategoryczne przekształcają się na format numeryczny odpowiedni dla większości algorytmów uczenia maszynowego. Pozwala to modelowi uczyć się zależności między poziomem zadowolenia a innymi cechami w danych.

1. **df_train['satisfaction'].value_counts():**
    - Ta linia oblicza liczbę wystąpień każdej unikatowej wartości w kolumnie "satisfaction". Wyświetla tabelę częstości pokazującą, ile wpisów należy do każdej kategorii (np. "1 niezadowolony", "2 neutralny" itd.).Może to być pomocne w zrozumieniu rozkładu poziomów zadowolenia w danych.
2. **encoded_satisfaction = df_train['satisfaction'].astype('category'):**
    - Ta linia konwertuje kolumnę "satisfaction" na typ danych kategorycznych. Ten krok jest ważny, ponieważ modele uczenia maszynowego zazwyczaj wymagają cech numerycznych. Konwertując kolumnę na typ kategoryczny, pandas rozpoznaje, że zawiera ona etykiety dyskretne, a nie tekst swobodny.
3. **df_train['satisfaction'] = encoded_satisfaction.cat.codes:**
    - Ta linia wykonuje rzeczywiste kodowanie. Używa atrybutu .cat kolumny kategorycznej (encoded_satisfaction) i właściwości .codes, aby przypisać każdej kategorii unikalny kod całkowity. Na przykład "1 niezadowolony" może zostać zakodowany jako 0, "2 neutralny" jako 1 itd. Ten proces nazywa się kodowaniem etykiet, gdzie każda kategoria jest mapowana na reprezentację numeryczną.
4. **print("Kody:", encoded_satisfaction.cat.categories):**
    - Ta linia po prostu drukuje etykiety (kategorie) wraz z ich odpowiadającymi kodami. Może to być pomocne w celach weryfikacyjnych, aby upewnić się, że kodowanie zostało wykonane zgodnie z oczekiwaniami.
5. **df_train.head():**
    - Ta linia wyświetla pierwsze kilka wierszy zbioru danych df_train po procesie kodowania. Wyświetla się teraz kolumna "satisfaction" zawierająca kody całkowite zamiast oryginalnych etykiet tekstowych.

Przygotowanie danych dla modelu

1. Definiowanie zmiennej cel (y):

- y = df_train["satisfaction"].copy():
    - Ta linia tworzy nową zmienną y i przypisuje do niej wartości z kolumny "satisfaction" z df_train.
    - Użycie copy() zapewnia, że tworzymy kopię danych, a nie modyfikujemy bezpośrednio kolumny w oryginalnej ramce danych.
    - Zmienna y będzie reprezentować klasę, którą chcemy przewidzieć (poziom zadowolenia).

2. Definiowanie cech wejściowych (X):

- X = df_train.drop(["satisfaction", "Unnamed: 0", "id"], axis=1).copy():
    - Ta linia tworzy nową zmienną X i przypisuje do niej kolumny z df_train z wyjątkiem:
        - "satisfaction" (ponieważ to właśnie chcemy przewidzieć).
        - "Unnamed: 0" (jeśli jest to zbędny indeks wiersza).
        - "id" (jeśli jest to unikalny identyfikator, który nie jest istotną cechą).
    - Użycie drop() usuwa te kolumny z kopii df_train.
    - Użycie copy() zapewnia, że tworzymy kopię danych, a nie modyfikujemy bezpośrednio oryginalnej ramki danych.
    - Zmienna X będzie zawierać cechy, które model będzie wykorzystywał do przewidywania zmiennej y.

3. Przygotowanie danych testowych (podobne kroki):

- X_df_test = df_test.drop(["satisfaction", "Unnamed: 0", "id"], axis=1).copy():
    - Ten krok jest analogiczny do poprzedniego, ale wykonuje się go dla danych testowych df_test.
    - Dane testowe zostaną wykorzystane później do oceny wydajności wytrenowanego modelu.

4. Uzupełnianie brakujących wartości (tylko dla jednej kolumny):

- df_train.isna().sum():
    - Ta linia wyświetla sumę brakujących wartości (NaN) w każdej kolumnie df_train.
    - Może to pomóc w zidentyfikowaniu kolumn wymagających uzupełnienia.
- missing = ['Arrival Delay in Minutes']:
    - Ta linia tworzy listę zawierającą nazwę kolumny ("Arrival Delay in Minutes"), w której występują brakujące wartości.
- Pętla for col in missing iteruje po elementach listy missing:
    - imputer_mean = SimpleImputer(strategy='mean', missing_values=np.nan):
        - Tworzony jest obiekt SimpleImputer z biblioteki scikit-learn.
        - Służy on do uzupełniania brakujących wartości strategią mean (średnią), a missing_values=np.nanokreśla, że brakujące wartości są reprezentowane przez NaN.
    - X[[col]] = imputer_mean.fit_transform(X[[col]]):
        - Ten fragment wykorzystuje obiekt imputer_mean do dopasowania (fit) i przekształcenia (transform) kolumny col w ramce X.
            - fit oblicza średnią wartości w kolumnie.
            - transform zastępuje brakujące wartości w kolumnie obliczoną średnią.
        - Podwójne kwadratowe nawiasy [[col]] zapewniają, że X traktowany jest jako dwuwymiarowa tablica,a nie seria, co jest wymagane przez imputer_mean.

5. Uzupełnianie brakujących wartości w danych testowych (podobne kroki):

- Kroki uzupełniania brakujących wartości są analogiczne do tych wykonanych dla X, ale dotyczą one X_df_test.

Kodowanie danych i skalowanie

Ten kod przetwarza dane, kodując zmienne kategoryczne i skalując dane numeryczne.

1. Sprawdzenie informacji o danych:

- X.info():
    - Ta linia wyświetla informacje o typach danych w X.
    - Pomaga to w identyfikacji kolumn wymagających kodowania.

2. Kodowanie zmiennych kategorycznych:

- enc = OneHotEncoder(handle_unknown='ignore'):
    - Tworzony jest obiekt OneHotEncoder z biblioteki scikit-learn.
    - Służy on do kodowania zmiennych kategorycznych w postaci one-hot encoding.
    - handle_unknown='ignore' oznacza, że jeśli napotka wartości nieznane podczas kodowania danych testowych, zostaną one zignorowane.
- Pętla iteruje po wybranych kolumnach zawierających dane kategoryczne ("Gender", "Customer Type", "Type of Travel", "Class"):
    - enc.fit(X[["col_name"]]):
        - Ta linia dopasowuje obiekt enc do danych w kolumnie col_name z X.
        - Podczas dopasowywania enc identyfikuje unikatowe kategorie występujące w tej kolumnie.
    - transformed_sleep = enc.transform(X[["col_name"]]).toarray():
        - Ta linia przekształca dane w kolumnie col_name z X za pomocą dopasowanego obiektu enc.
        - Wynik (transformed_sleep) jest tablicą NumPy, a toarray() konwertuje ją do zwykłej tablicy.
        - Kodowanie one-hot tworzy nową kolumnę dla każdej unikalnej kategorii, a wartość 1 wskazuje przynależność do danej kategorii, 0 oznacza brak przynależności.
    - col_df = pd.DataFrame(transformed_sleep, columns=enc.get_feature_names_out(["col_name"])):
        - Ta linia tworzy nową ramkę danych col_df z zakodowanymi danymi (transformed_sleep).
        - Nazwy kolumn w col_df są tworzone przy użyciu metody get_feature_names_out, która zwraca nazwy nowych kolumn one-hot na podstawie oryginalnej nazwy kolumny (col_name).
    - Podobne kroki powtarzane są dla kolejnych kolumn kategorycznych.

3. Kodowanie dla danych testowych (podobne kroki):

- Kroki kodowania one-hot dla danych testowych X_df_test są analogiczne do tych wykonanych dla X.

4. Skalowanie danych numerycznych:

- columns_to_scale:

    - Ta linia tworzy listę zawierającą nazwy kolumn z danymi numerycznymi, które powinny zostać przeskalowane.
- Pętla iteruje po kolumnach z listy columns_to_scale:
    - scaler = MinMaxScaler():
        - Tworzony jest obiekt MinMaxScaler z biblioteki scikit-learn.
        - Służy on do skalowania danych numerycznych do zakresu [0, 1].
    - X[column] = scaler.fit_transform(X[[column]]):
        - Ta linia dopasowuje (fit) obiekt scaler do danych w kolumnie column z X (oblicza wartości min i max dla skalowania).
        - Następnie przekształca (transform) dane w tej kolumnie, skalując je do zakresu [0, 1].

5. Skalowanie danych testowych:

- Kroki skalowania danych numerycznych dla X_df_test są analogiczne do tych wykonanych dla X.

6. Podgląd danych po przetworzeniu:

- df_train.head():
    - Ta linia wyświetla pierwsze kilka wierszy df_train po przetworzeniu.
    - Powinny teraz zawierać zakodowane kolumny kategoryczne i przeskalowane dane numeryczne.

Analiza kodu: Łączenie danych po przetworzeniu

Ten kod łączy ze sobą przetworzone dane (cechy wejściowe) w celu przygotowania ich do etapu uczenia modelu:

1. Łączenie danych treningowych:

- X_transformed = X.copy():
    - Tworzy się kopię X i przypisuje się ją do nowej zmiennej X_transformed.
- X_transformed = X_transformed.drop(["Gender","Class", "Type of Travel", "Customer Type"],axis=1):
    - Ta linia usuwa z X_transformed kolumny, które zostały już zakodowane i nie są potrzebne jako osobne cechy (Gender, Class, Type of Travel, Customer Type).
- X_transformed = pd.concat([X_transformed, class_df, type_travel_df, customer_df, gender_df], axis=1):
    - Ta linia łączy ze sobą kilka ramek danych:
        - X_transformed (zawierającą oryginalne cechy numeryczne).
        - class_df, type_travel_df, customer_df, gender_df (zawierające zakodowane dane kategoryczne).
    - axis=1 oznacza łączenie kolumn (pionowo).
    - W efekcie powstaje X_transformed zawierająca zarówno oryginalne cechy numeryczne, jak i zakodowane cechy kategoryczne.

2. Łączenie danych testowych (podobne kroki):

- Kroki łączenia danych dla zestawu testowego X_df_test są analogiczne do tych wykonanych dla X_transformed.

3. Podgląd połączonych danych:

- X_transformed.head():
    - Ta linia wyświetla pierwsze kilka wierszy X_transformed po połączeniu, aby sprawdzić, czy operacja przebiegła pomyślnie. Powinna teraz zawierać wszystkie przygotowane cechy wejściowe.
    - Podobnie X_df_test_transformed.head() dla danych testowych.

Łączenie danych po przetworzeniu

Ten kod łączy ze sobą przetworzone dane (cechy wejściowe) w celu przygotowania ich do etapu uczenia modelu.

1. Łączenie danych treningowych:

- X_transformed = X.copy():
    - Tworzy się kopię X i przypisuje się ją do nowej zmiennej X_transformed.
- X_transformed = X_transformed.drop(["Gender","Class", "Type of Travel", "Customer Type"],axis=1):
    - Ta linia usuwa z X_transformed kolumny, które zostały już zakodowane i nie są potrzebne jako osobne cechy (Gender, Class, Type of Travel, Customer Type).
- X_transformed = pd.concat([X_transformed, class_df, type_travel_df, customer_df, gender_df], axis=1):
    - Ta linia łączy ze sobą kilka ramek danych:
        - X_transformed (zawierającą oryginalne cechy numeryczne).
        - class_df, type_travel_df, customer_df, gender_df (zawierające zakodowane dane kategoryczne).
    - axis=1 oznacza łączenie kolumn (pionowo).
    - W efekcie powstaje X_transformed zawierająca zarówno oryginalne cechy numeryczne, jak i zakodowane cechy kategoryczne.

2. Łączenie danych testowych (podobne kroki):

- Kroki łączenia danych dla zestawu testowego X_df_test są analogiczne do tych wykonanych dla X_transformed.

3. Podgląd połączonych danych:

- X_transformed.head():

    - Ta linia wyświetla pierwsze kilka wierszy X_transformed po połączeniu, aby sprawdzić, czy operacja przebiegła pomyślnie.
    - Podobnie X_df_test_transformed.head() dla danych testowych.

Trening modeli, Ewaluacja i Tuning

1. Trening modeli:

- train_test_split:
    - Dzieli dane na zbiór treningowy (X_train, y_train) i testowy (X_test, y_test) stosując losowy podział z wielkością zbioru testowego test_size=0.3 i ziarnem random_state=42 dla zapewnienia reprodukowalności wyników.
- Tworzy i trenuje różne modele klasyfikacyjne:
    - RandomForestClassifier
    - SVC (Support Vector Classifier)
    - GaussianNB (Naive Bayes)
    - KNeighborsClassifier
    - LogisticRegression
    - Każdy model jest trenowany na danych treningowych X_train i y_train.
    - Parametr random_state=0 zapewnia stałe inicjowanie dla każdego modelu.
    - W przypadku LogisticRegression zwiększono limit iteracji (max_iter=1000) ze względu na potencjalny błąd domyślnego limitu.

2. Przewidywanie i ewaluacja:

- predict przewiduje klasy dla danych testowych za pomocą wytrenowanych modeli.
- predict_proba (tylko dla LogisticRegression) zwraca prawdopodobieństwo przynależności do każdej klasy.
- Oblicza wskaźniki oceny:
    - recall_score - odsetek trafnie pozytywnych klasyfikacji.
    - precision_score - odsetek pozytywnych klasyfikacji, które były poprawne.
    - f1_score - średnia harmoniczna precision i recall.
- Wyświetla wyniki dla każdego modelu.
- Dodatkowo dla LogisticRegression drukuje się:
    - Pierwsze elementy y_pred_lr_prob i y_pred_lr (prawdopodobieństwa i klasy przewidywane).
    - Wynik score (accuracy) na danych testowych.
    - Współczynniki modelu (coef_).
    - Macierz confusion matrix.
    - Wizualizacja macierzy confusion matrix za pomocą ConfusionMatrixDisplay.
    - Wykres ROC AUC (Area Under the Curve).
    - Wyniki cross_val_score dla 10-krotnej walidacji krzyżowej.

3. Tuning hiperparametrów:

- GridSearchCV służy do wyszukiwania najlepszych parametrów dla każdego modelu.
    - Definiuje się siatkę parametrów do sprawdzenia (grid).
    - Uruchomienie GridSearchCV.fit przeprowadza trening modelu z różnymi kombinacjami parametrów z siatki na danych treningowych i wybiera zestaw parametrów, który osiąga najlepszą wydajność na podstawie walidacji krzyżowej.
- Wyszukiwanie najlepszych parametrów przeprowadza się dla:

    - RandomForestClassifier
    - SVC
    - KNeighborsClassifier
- Wyświetla najlepsze znalezione parametry (best_params_) i dokładność (best_score_) dla każdego z tych modeli.
- Na podstawie wyników wyszukiwania tworzy się nowe modele z najlepszymi parametrami:
    - rfc_1, svc_1, knn_1
- Ponownie trenuje się te modele na danych treningowych.
- Podobnie jak wcześniej, dokonuje się przewidywania na danych testowych i ocenia się wydajność modeli za pomocą wskaźników recall_score, precision_score, f1_score.
- Wykorzystuje się również 10-krotną walidację krzyżową (cross_val_score) do oszacowania stabilności wyników.

Ten kod pozwala na porównanie różnych modeli uczenia maszynowego pod kątem ich skuteczności w zadaniu klasyfikacji. Ewaluacja obejmuje wskaźniki recall, precision, f1-score, a także wizualizacje macierzy confusion matrix i krzywej ROC AUC. Dodatkowo przeprowadza się tuning hiperparametrów w celu poprawy wydajności modeli.