Klasyfikacja binarna obecności osób w pomieszczeniu na podstawie danych sensorowych (Room Occupancy Estimation)

- Michał Adamiec 217633
- Piotr Lewandowski 217357

Streszczenie

Celem projektu jest opracowaneie modelu klasyfikacyjnego umożliwiającego wykrycie osób w pomieszczeniu na podstawie danych wielu nieinwazyjnych czujników środowiskowych, takich jak temperatura, światło, dźwięk, CO2 oraz PIR (pasywny czujnik podczerwieni). Zmienna docelowa została przekształcona do postaci binarnej, co pozwala na rozróżnienie sytuacji, gdy pomieszczenie jest puste lub zajęte.

Słowa kluczowe

- klasyfikacja binarna
- analiza danych
- czujniki środowiskowe
- obecność w pomieszczeniu
- uczenie maszynowe

Wprowadzenie

Projekt dotyczy wykrywania obecności osób w pomieszczeniu na podstawie danych z czujników środowiskowych.

Przedmiot badania

Analizujemy dane zebrane w pomieszczeniu wyposażonym w 7 czujników, rejestrujących parametry środowiskowe co 30 sekund.

Cel projketu

Celem jest stworzenie modelu klasyfikacyjnego, który na podstawie danych z czujników przewidzi obecność osób w pomieszczeniu (klasyfikacja binarna).

Wstępna analiza danych

Charakterystyka zbioru danych

- Liczba obserwacji: 10129
- Liczba cech: 18
- Typ cech: numeryczne (real)

Przedstawienie dostępnych zmiennych

- **Date** - data pomiaru w formacie YYYY/MM/DD
- **Time** - godzina pomiaru w formacie HH:MM:SS
- **S1_Temp, S2_Temp, S3_Temp, S4_Temp** - temperatura z czujników S1-S4 (w stopniach Celsjusza)
- **S1_Light, S2_Light, S3_Light, S4_Light** - natężenie światła z czujników S1-S4 (w luksach)
- **S1_Sound, S2_Sound, S3_Sound, S4_Sound** - poziom dźwięku z czujników S1-S4 (w woltach, odczyt z ADC)
- **S5_CO2** - stężenie CO2 z czujnika S5 (w PPM)
- **S5_CO2_Slope** - nachylenie zmian CO2 w oknie czasowym
- **S6_PIR** - detekcja ruchu przez czujnik PIR S6
  - *0* - brak ruchu
  - *1* - wykryto ruch
- **S7_PIR** - detekcja ruchu przez czujnik PIR S7
  - *0* - brak ruchu
  - *1* - wykryto ruch
- **Room_Occupancy_Count** - liczba osób w pomieszczeniu (zmienna docelowa)
  - *0* - pomieszczenie puste
  - *1, 2, 3* - liczba obecnych osób

Jeśli nie ma odpowiednich pakietów

In [1]:
```
%pip install -q scikit-learn ucimlrepo seaborn matplotlib
```

```
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 24.0 -> 25.3
[notice] To update, run: C:\Users\jlewa\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip
```

Importy

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# function that fetches dataset from ucimlrepo
from ucimlrepo import fetch_ucirepo
#Data manipulation
from sklearn.model_selection import train_test_split
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
#KNN
from sklearn.neighbors import KNeighborsClassifier
#Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score, roc_auc_score, confusi
```

Pobranie zbioru danych i informacje o zmiennych

In [3]:
```
# fetch dataset
room_occupancy_estimation = fetch_ucirepo(id=864)

# data (as pandas dataframes)
X = room_occupancy_estimation.data.features
y = room_occupancy_estimation.data.targets

# metadata
# print(room_occupancy_estimation.metadata)

# variable information
print(room_occupancy_estimation.variables)
```

```
                   name     role        type demographic  \
0                  Date  Feature        Date        None
1                  Time  Feature        Date        None
2               S1_Temp  Feature  Continuous        None
3               S2_Temp  Feature  Continuous        None
4               S3_Temp  Feature  Continuous        None
5               S4_Temp  Feature  Continuous        None
6              S1_Light  Feature     Integer        None
7              S2_Light  Feature     Integer        None
8              S3_Light  Feature     Integer        None
9              S4_Light  Feature     Integer        None
10             S1_Sound  Feature  Continuous        None
11             S2_Sound  Feature  Continuous        None
12             S3_Sound  Feature  Continuous        None
13             S4_Sound  Feature  Continuous        None
14               S5_CO2  Feature     Integer        None
15        S5_CO2_Slope  Feature  Continuous        None
16               S6_PIR  Feature      Binary        None
17               S7_PIR  Feature     Integer        None
18  Room_Occupancy_Count   Target     Integer        None

                                    description       units missing_values
0                                          None  YYYY/MM/DD             no
1                                          None    HH:MM:SS             no
2                                          None           C             no
3                                          None           C             no
4                                          None           C             no
5                                          None           C             no
6                                          None         Lux             no
7                                          None         Lux             no
8                                          None         Lux             no
9                                          None         Lux             no
10                 amplifier output read by ADC       Volts             no
11                 amplifier output read by ADC       Volts             no
12                 amplifier output read by ADC       Volts             no
13                 amplifier output read by ADC       Volts             no
14                                         None         PPM             no
15  Slope of CO2 values taken in a sliding window        None             no
16       Binary value conveying motion detection        None             no
17       Binary value conveying motion detection        None             no
18                                Ground Truth        None             no
```

Podgląd danych

In [4]:
```
X.head()
```

Out[4]:

	Date	Time	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3
0	2017/12/22	10:49:41	24.94	24.75	24.56	25.38	121	34	
1	2017/12/22	10:50:12	24.94	24.75	24.56	25.44	121	33	
2	2017/12/22	10:50:42	25.00	24.75	24.50	25.44	121	34	
3	2017/12/22	10:51:13	25.00	24.75	24.56	25.44	121	34	
4	2017/12/22	10:51:44	25.00	24.75	24.56	25.44	121	34	

In [5]:
```
y.head()
```

Out[5]:

	Room_Occupancy_Count
0	1
1	1
2	1
3	1
4	1

Pierwsze wystąpienia dla każdej wartości Room_Occupancy_Count

In [6]:
```
unique_values = y['Room_Occupancy_Count'].unique()
indices = [y[y['Room_Occupancy_Count'] == val].index[0] for val in unique_values
display(X.loc[indices])
```

	Date	Time	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3
0	2017/12/22	10:49:41	24.94	24.75	24.56	25.38	121	34	
75	2017/12/22	11:28:29	25.38	25.44	24.81	25.69	150	187	
191	2017/12/22	12:30:16	25.69	28.44	25.19	26.00	156	244	
346	2017/12/22	14:15:58	26.19	27.25	26.13	26.44	19	22	

In [7]:
```
display(y.loc[indices])
```

	Room_Occupancy_Count
0	1
75	2
191	3
346	0

Liczba wystąpień każdej zmiennej Room_Occupancy_Count w zbiorze danych

In [8]:
```
print(y['Room_Occupancy_Count'].value_counts())
```

```
Room_Occupancy_Count
0    8228
2     748
3     694
1     459
Name: count, dtype: int64
```

Tworzenie nowych cech

Nowo utworzone cechy

- **Temp_mean** – średnia temperatura z czujników S1–S4 (w stopniach Celsjusza)
- **Light_sum** - suma natężenia światła z czterech czujników (w luksach)
- **Sound_mean** - średnia wartość poziomu dźwięku z czterech czujników (w woltach)
- **CO2_to_Sound** - stosunek stężenia CO2 do sumy poziomu dźwięku (bezwymiarowe)
- **PIR_active** - liczba aktywnych czujników ruchu PIR (wartość 0, 1 lub 2)

In [9]:
```
X['Temp_mean'] = X[['S1_Temp', 'S2_Temp', 'S3_Temp', 'S4_Temp']].mean(axis=1)
X['Light_sum'] = X[['S1_Light', 'S2_Light', 'S3_Light', 'S4_Light']].sum(axis=1)
X['Sound_mean'] = X[['S1_Sound', 'S2_Sound', 'S3_Sound', 'S4_Sound']].mean(axis=
X['CO2_to_Sound'] = X['S5_CO2'] / (X[['S1_Sound', 'S2_Sound', 'S3_Sound', 'S4_Sc
X['PIR_active'] = X['S6_PIR'] + X['S7_PIR']

X.head()
```

Out[9]:

	Date	Time	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3
0	2017/12/22	10:49:41	24.94	24.75	24.56	25.38	121	34	
1	2017/12/22	10:50:12	24.94	24.75	24.56	25.44	121	33	
2	2017/12/22	10:50:42	25.00	24.75	24.50	25.44	121	34	
3	2017/12/22	10:51:13	25.00	24.75	24.56	25.44	121	34	
4	2017/12/22	10:51:44	25.00	24.75	24.56	25.44	121	34	

5 rows × 23 columns

Przygotowanie danych do klasyfikacji binarnej

Binarna zmienna docelowa i wybór cech

In [10]:
```
#Klasyfikacja binarna: 0 - pusty pokój, 1 - zajęty pokój (przekształcenie y)
y = y.rename(columns={'Room_Occupancy_Count': 'Occupied'})
y['Occupied'] = (y['Occupied'] == 3).astype(int)
y = y[['Occupied']]
y.head()

#Usunięcie kolumn Date i Time z X
X=X.drop(columns=['Date', 'Time'])
```

Pierwsze wystąpienia dla każdej wartości Occupied

In [11]:
```
unique_values = y['Occupied'].unique()
indices = [y[y['Occupied'] == val].index[0] for val in unique_values]
display(X.loc[indices])
```

	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_Light	S4_Light	S1_
0	24.94	24.75	24.56	25.38	121	34	53	40	
191	25.69	28.44	25.19	26.00	156	244	190	64	

2 rows × 21 columns

In [12]:
```
display(y.loc[indices])
```

	Occupied
0	0
191	1

Liczba wystąpień każdej zmiennej Occupied w zbiorze danych

In [13]:
```
print(y['Occupied'].value_counts())
```

```
Occupied
0    9435
1     694
Name: count, dtype: int64
```

Podział na zbiory treningowy i walidacyjny (60/40)

Dane zostały podzielone na zbiór treningowy (60%) i walidacyjny (40%), aby rzetelnie ocenić skuteczność modeli na nowych, niewidzianych wcześniej danych.

In [14]:
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.4, random_st

print("Liczba rekordów w zbiorze treningowym:", len(X_train))
print("Liczba rekordów w zbiorze walidacyjnym:", len(X_val))
```

```
Liczba rekordów w zbiorze treningowym: 6077
Liczba rekordów w zbiorze walidacyjnym: 4052
```

Metody klasyfikacji danych

Drzewo decyzyjne

Drzewo decyzyjne to narzędzie, wspierające proces podejmowania decyzji. Polega na przedstawieniu graficznym różnorodnych opcji i ich możliwych konsekwencji - wizualizacja zwykle przyjmuje postać rozgałęziającego się diagramu, co nawiązuje do struktury drzewa.

In [15]:
```
# Dopasowanie modelu
dt_model = DecisionTreeClassifier(max_depth=2, random_state=1)
dt_model.fit(X_train, y_train)

# Predykcja na zbiorze walidacyjnym
y_pred_dt = dt_model.predict(X_val)
y_proba_dt = dt_model.predict_proba(X_val)[:, 1]

print(y_pred_dt)
print(y_proba_dt)
```

```
[0 0 0 ... 0 0 0]
[0.01003944 0.01003944 0.01003944 ... 0.01003944 0.01003944 0.01003944]
```

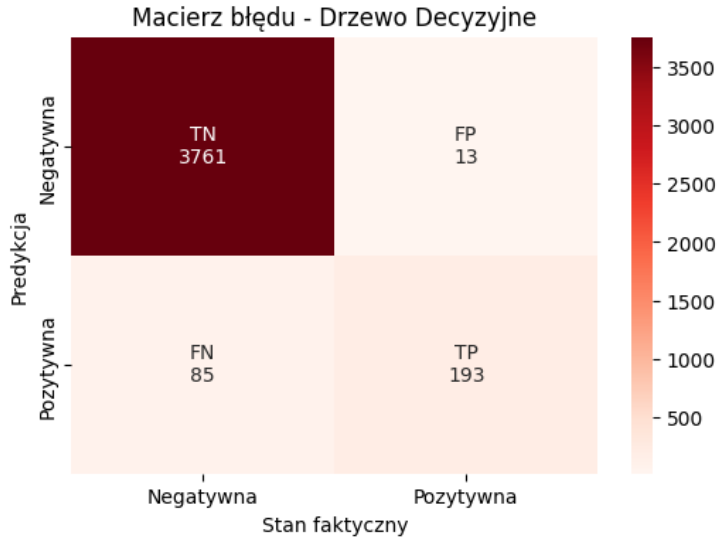
Macierze pomyłek i współczynniki jakości klasyfikacji

Klasyfikacja negatywna oznacza niewystępowanie zjawiska (wartość 0 - pokój pusty "No"), natomiast klasyfikacja pozytywna oznacza wystąpienie zjawiska (wartość 1 - pokój zajęty "Yes"). Wyjaśnienie oznaczeń:

- TN - prawidłowa klasyfikacja negatywna
- TP - prawidłowa klasyfikacja pozytywna
- FP - fałszywa klasyfikacja poztywna
- FN - fałszywa klasyfikacja negatywna

$TN + TP + FP + FN =$ ilość rekordów testowych

In [16]:
```
# Macierz pomyłek
cm_dt = confusion_matrix(y_val, y_pred_dt)
dt_tn, dt_fp, dt_fn, dt_tp = cm_dt.ravel()

# Dane i etykiety do wizualizacji
confusion_matrix_data = [
    [dt_tn, dt_fp],
    [dt_fn, dt_tp]
]
confusion_matrix_labels = [
    [f"TN\n{dt_tn}", f"FP\n{dt_fp}"],
    [f"FN\n{dt_fn}", f"TP\n{dt_tp}"]
]

plt.figure(figsize=(6, 4))
ax = sns.heatmap(confusion_matrix_data, annot=confusion_matrix_labels, fmt="", c
            xticklabels=["Negatywna", "Pozytywna"],
            yticklabels=["Negatywna", "Pozytywna"])
plt.title("Macierz błędu - Drzewo Decyzyjne")
plt.xlabel("Stan faktyczny")
plt.ylabel("Predykcja")
plt.show()
```



Współczynniki wydajności klasyfikacji

Trafność (Accuracy) - odsetek poprawnych, prawidłowych klasyfikacji dokonywanych przez model

$$\frac{TP+TN}{TP+TN+FP+FN}$$

Czułość (Recall) - zdolność modelu do wykrywania rzeczywistych pozytywnych przykładów

$$\frac{TP}{TP+FN}$$

Specyficzność (Specificity) – zdolność modelu do prawidłowego wykrywania przykładów negatywnych (czyli poprawnego rozpoznania klasy 0, np. pustego pokoju)

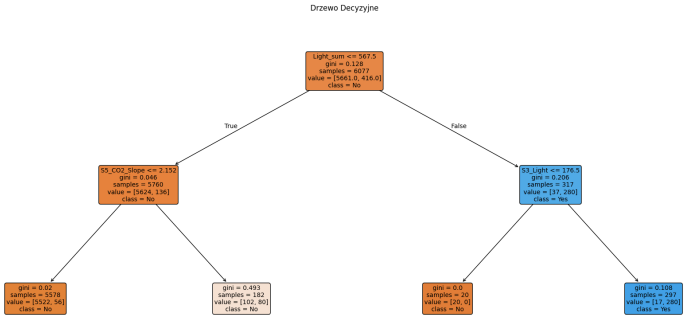
$$\frac{TN}{TN+FP}$$

AUC (Area Under the Curve) - to miara używana do oceny wydajności modelu klasyfikacji binarnej na podstawie krzywej ROC.

In [17]:
```
accuracy_dt = accuracy_score(y_val, y_pred_dt)
recall_dt = recall_score(y_val, y_pred_dt)
specifity_dt = recall_score(y_val, y_pred_dt, pos_label=0)
auc_dt = roc_auc_score(y_val, y_proba_dt)

print("Accuracy modelu drzewa decyzyjnego:", accuracy_dt)
print("Recall modelu drzewa decyzyjnego:", recall_dt)
print("Specifity modelu drzewa decyzyjnego:", specifity_dt)
print("AUC modelu drzewa decyzyjnego:", auc_dt)
```

```
Accuracy modelu drzewa decyzyjnego: 0.9758144126357354
Recall modelu drzewa decyzyjnego: 0.6942446043165468
Specifity modelu drzewa decyzyjnego: 0.9965553789083201
AUC modelu drzewa decyzyjnego: 0.9183332189574255
```

In [18]:
```
plt.figure(figsize=(24, 10))
plot_tree(dt_model, filled=True, feature_names=X_train.columns, class_names=['No
plt.title("Drzewo Decyzyjne")
plt.show()
```

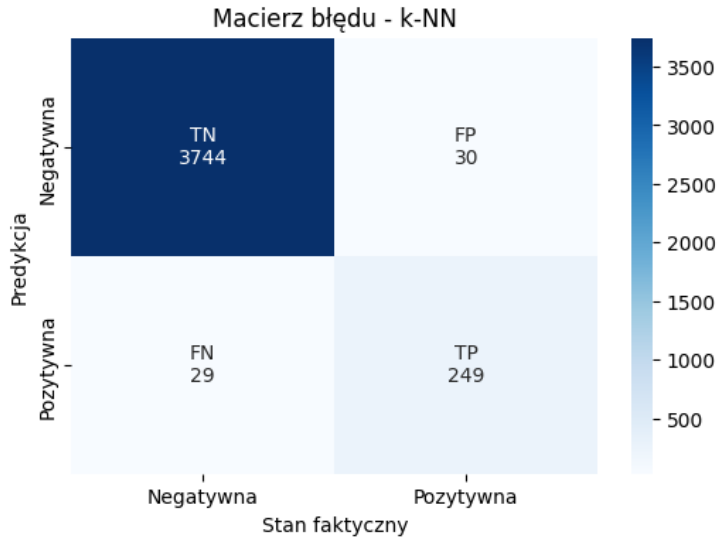


KNN - algorytm najbliższego sąsiada

To jeden z najbardziej podstawowych, ale niezbędnych algorytmów klasyfikacyjnych w uczeniu maszynowym. Należy do domeny uczenia się nadzorowanego i znajduje szerokie zastosowanie w rozpoznawaniu wzorców, eksploracji danych i wykrywaniu włamań.

In [19]:
```
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_val_scaled = scaler.transform(X_val)

# Stworzenie i dopasowanie modelu 48-NN
knn_model = KNeighborsClassifier(n_neighbors=20, weights='uniform', algorithm='a
knn_model.fit(X_train_scaled, y_train.values.ravel())

# Dokonaj predykcji na danych testowych
y_pred_knn = knn_model.predict(X_val_scaled)
y_proba_knn = knn_model.predict_proba(X_val_scaled)[:, 1]
```

```
# Klasyfikator 1-NN
knn_cm = confusion_matrix(y_val, y_pred_knn)
knn_tn, knn_fp, knn_fn, knn_tp = knn_cm.ravel()

# Macierz pomyłek
confusion_matrix_data = [
    [knn_tn, knn_fp],
    [knn_fn, knn_tp]
]

# Etykiety dla macierzy pomyłek
confusion_matrix_labels = [
    [f"TN\n{knn_tn}", f"FP\n{knn_fp}"],
    [f"FN\n{knn_fn}", f"TP\n{knn_tp}"]
]

# Rysowanie macierzy pomyłek
plt.figure(figsize=(6, 4))
ax = sns.heatmap(confusion_matrix_data, annot=confusion_matrix_labels, fmt="", c
            xticklabels=["Negatywna", "Pozytywna"],
            yticklabels=["Negatywna", "Pozytywna"])

plt.title("Macierz błędu - k-NN")
plt.xlabel("Stan faktyczny")
plt.ylabel("Predykcja")
plt.show()
```



In [20]:
```
accuracy_knn = accuracy_score(y_val, y_pred_knn)
recall_knn = recall_score(y_val, y_pred_knn)
specifity_knn = recall_score(y_val, y_pred_knn, pos_label=0)
auc_knn = roc_auc_score(y_val, y_proba_knn)

print("Accuracy modelu k-NN:", accuracy_knn)
print("Recall modelu k-NN:", recall_knn)
```

```
print("Specifity modelu k-NN:", specifity_knn)
print("AUC modelu k-NN:", auc_knn)
```

```
Accuracy modelu k-NN: 0.9854392892398816
Recall modelu k-NN: 0.89568345323741
Specifity modelu k-NN: 0.9920508744038156
AUC modelu k-NN: 0.9979917496845131
```

Las Losowy (Random Forest)

Las losowy to zaawansowana metoda zespołowa (ang. ensemble method), stanowiąca naturalne rozwinięcie koncepcji pojedynczego drzewa decyzyjnego. Algorytm ten polega na wygenerowaniu dużej liczby drzew decyzyjnych, które uczą się niezależnie od siebie. Ostateczny wynik klasyfikacji ustalany jest na drodze „głosowania" – system wybiera tę decyzję, która została wskazana przez większość drzew w lesie. Dzięki temu podejściu model jest bardziej stabilny i precyzyjny niż pojedyncze drzewo.

Optymalizacja liczby drzew (`n_estimators`)

Poniższy fragment kodu przeprowadza eksperyment mający na celu dobranie optymalnej liczby drzew w lesie losowym. Zdefiniowano listę wartości od **10 do 1000**, a następnie w pętli trenowano kolejne modele.

Cel analizy:

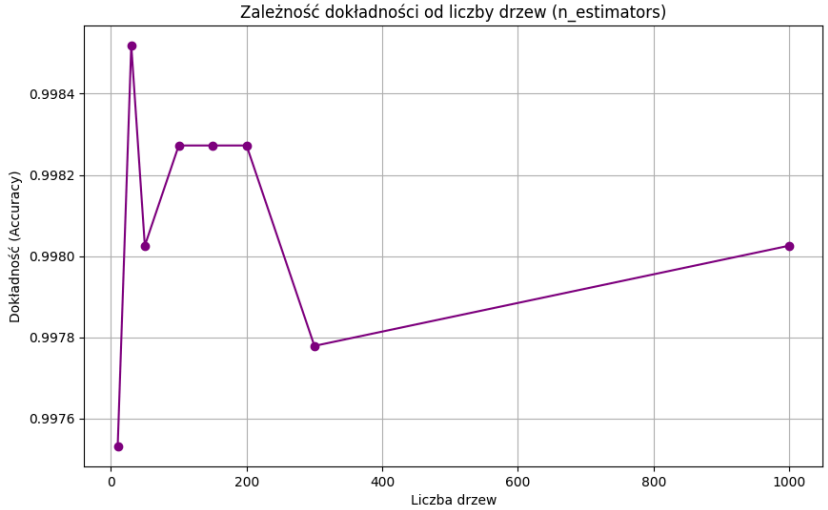
- Znalezienie balansu między jakością modelu a czasem obliczeń.
- Identyfikacja punktu stabilizacji, w którym dodawanie kolejnych drzew przestaje istotnie poprawiać wyniki.

Metodyka:

1. Dla każdej iteracji mierzona jest dokładność klasyfikacji (**accuracy**) na zbiorze walidacyjnym.
2. Wyniki są wizualizowane na wykresie liniowym, aby ułatwić interpretację.

In [21]:
```
# Lista liczby drzew do sprawdzenia
n_trees_list = [10, 30, 50, 100, 150, 200, 300, 1000]
scores = []

for n in n_trees_list:
    rf = RandomForestClassifier(n_estimators=n, random_state=42, n_jobs=-1)
    rf.fit(X_train, y_train.values.ravel())

    score = accuracy_score(y_val, rf.predict(X_val))
    scores.append(score)
    print(f"Drzew: {n} -> Dokładność: {score:.4f}")

n_best = n_trees_list[scores.index(max(scores))]
print(f'Najlepszy wynik dla {n_best} drzew')

plt.figure(figsize=(10, 6))
plt.plot(n_trees_list, scores, marker='o', linestyle='-', color='purple')
plt.title('Zależność dokładności od liczby drzew (n_estimators)')
plt.xlabel('Liczba drzew')
plt.ylabel('Dokładność (Accuracy)')
plt.grid(True)
plt.show()
```

```
Drzew: 10 -> Dokładność: 0.9975
Drzew: 30 -> Dokładność: 0.9985
Drzew: 50 -> Dokładność: 0.9980
Drzew: 100 -> Dokładność: 0.9983
Drzew: 150 -> Dokładność: 0.9983
Drzew: 200 -> Dokładność: 0.9983
Drzew: 300 -> Dokładność: 0.9978
Drzew: 1000 -> Dokładność: 0.9980
Najlepszy wynik dla 30 drzew
```

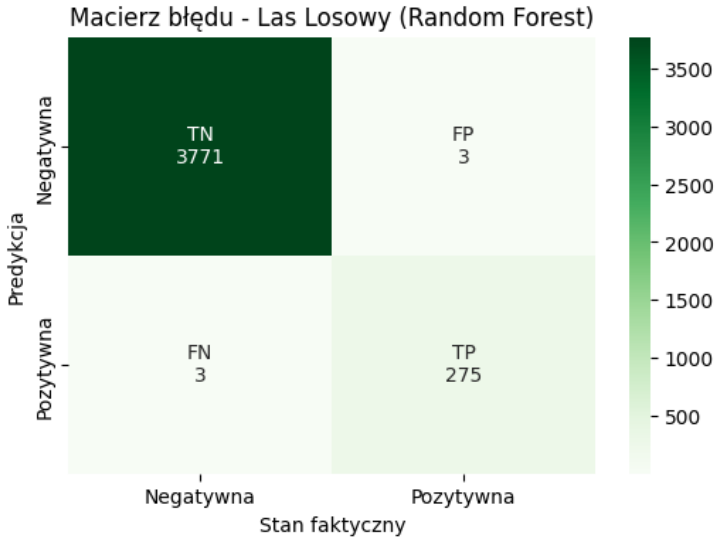


Trenowanie Finalnego Modelu i Analiza Błędów

Mając wyznaczoną optymalną liczbę drzew (`n_best`), przechodzimy do etapu **trenowania ostatecznego modelu** oraz szczegółowej oceny jego skuteczności.

In [22]:
```
# Stworzenie i dopasowanie modelu Lasu Losowego
# n_estimators to liczba drzew w lesie
rf_model = RandomForestClassifier(n_estimators=n_best, random_state=42)
rf_model.fit(X_train, y_train.values.ravel())

# Dokonaj predykcji na danych walidacyjnych
y_pred_rf = rf_model.predict(X_val)
y_proba_rf = rf_model.predict_proba(X_val)[:, 1]

# Obliczanie macierzy pomyłek dla Lasu Losowego
rf_cm = confusion_matrix(y_val, y_pred_rf)
rf_tn, rf_fp, rf_fn, rf_tp = rf_cm.ravel()

# Macierz pomyłek - dane
confusion_matrix_data_rf = [
    [rf_tn, rf_fp],
    [rf_fn, rf_tp]
]
```

```
# Etykiety dla macierzy pomyłek
confusion_matrix_labels_rf = [
    [f"TN\n{rf_tn}", f"FP\n{rf_fp}"],
    [f"FN\n{rf_fn}", f"TP\n{rf_tp}"]
]

# Rysowanie macierzy pomyłek
plt.figure(figsize=(6, 4))
ax = sns.heatmap(confusion_matrix_data_rf, annot=confusion_matrix_labels_rf, fmt
            xticklabels=["Negatywna", "Pozytywna"],
            yticklabels=["Negatywna", "Pozytywna"])

plt.title("Macierz błędu - Las Losowy (Random Forest)")
plt.xlabel("Stan faktyczny")
plt.ylabel("Predykcja")
plt.show()
```



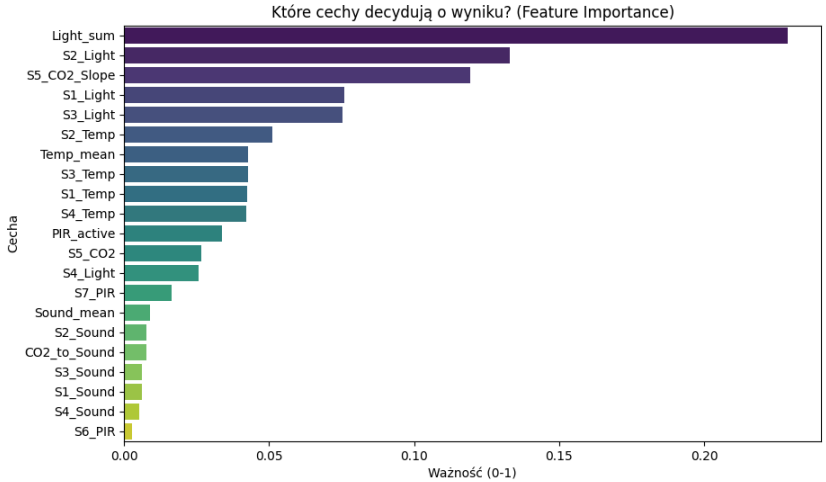
In [23]:
```
accuracy_rf = accuracy_score(y_val, y_pred_rf)
recall_rf = recall_score(y_val, y_pred_rf)
specifity_rf = recall_score(y_val, y_pred_rf, pos_label=0)
auc_rf = roc_auc_score(y_val, y_proba_rf)

print("Accuracy modelu lasu losowego:", accuracy_rf)
print("Recall modelu lasu losowego:", recall_rf)
print("Specifity modelu lasu losowego:", specifity_rf)
print("AUC modelu Lasu losowego:", auc_rf)
```

```
Accuracy modelu lasu losowego: 0.9985192497532083
Recall modelu lasu losowego: 0.9892086330935251
Specifity modelu lasu losowego: 0.9992050874403816
AUC modelu Lasu losowego: 0.9999447183112016
```

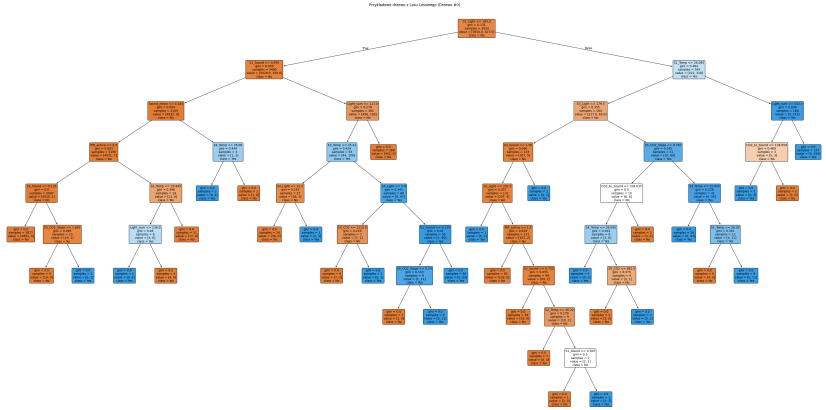
Analiza Ważności Cech

Jedną z zalet lasów losowych jest wbudowana możliwość oceny, które zmienne (cechy) mają największy wpływ na decyzje modelu.

In [24]:
```
import pandas as pd
# Pobieramy ważność cech z wytrenowanego modelu
importances = rf_model.feature_importances_
feature_names = X_train.columns

# Tworzymy ramkę danych do wykresu
forest_importances = pd.Series(importances, index=feature_names).sort_values(asc

# Rysujemy wykres
plt.figure(figsize=(10, 6))
sns.barplot(x=forest_importances.values, y=forest_importances.index, palette="vi

plt.title("Które cechy decydują o wyniku? (Feature Importance)")
plt.xlabel("Ważność (0-1)")
plt.ylabel("Cecha")
plt.show()
```



In [25]:
```
#przykładowe drzewo z lasu

one_tree_from_forest = rf_model.estimators_[0]

plt.figure(figsize=(50, 25))

plot_tree(one_tree_from_forest,
          filled=True,
          feature_names=X_train.columns,
          class_names=['No', 'Yes'],
          rounded=True,
          fontsize=10)

plt.title("Przykładowe drzewo z Lasu Losowego (Drzewo #0)")
plt.show()
```

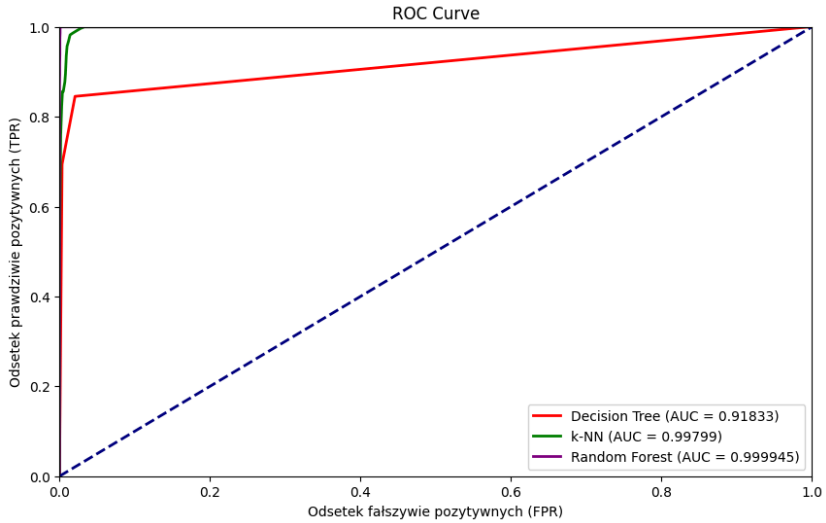


Krzywa ROC (Receiver Operating Characteristic)

Służy do oceny poprawności klasyfikatora. Przedstawia ona łączny opis jego czułości i specyficzności.

- $TPR = \frac{TP}{TP+FN}$ - czułość
- $FPR = \frac{TN}{TN+FP}$ - specyficzność

In [26]:
```
fpr_dt, tpr_dt, _ = roc_curve(y_val, y_proba_dt)
fpr_knn, tpr_knn, _ = roc_curve(y_val, y_proba_knn)
fpr_rf, tpr_rf, _ = roc_curve(y_val, y_proba_rf)

plt.figure(figsize=(10, 6))

# Krzywa ROC dla modelu drzewa decyzyjnego
plt.plot(fpr_dt, tpr_dt, color='red', lw=2, label=f'Decision Tree (AUC = {auc_dt
# Krzywa ROC dla modelu 1-NN
plt.plot(fpr_knn, tpr_knn, color='green', lw=2, label=f'k-NN (AUC = {auc_knn:.5f
# Krzywa ROC dla modelu lasu losowego
plt.plot(fpr_rf, tpr_rf, color='purple', lw=2, label=f'Random Forest (AUC = {auc

# Linia bazowa
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

# Ustawienia osi i tytułu
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Odsetek fałszywie pozytywnych (FPR)')
plt.ylabel('Odsetek prawdziwie pozytywnych (TPR)')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



Oto jak interpretować różne części tej krzywej:

- Punkt (0,0): Model klasyfikuje wszystkie próbki jako negatywne, co oznacza, że zarówno FPR, jak i TPR wynoszą 0
- Punkt (1,1): Model klasyfikuje wszystkie próbki jako pozytywne, co oznacza, że zarówno FPR, jak i TPR wynoszą 1
- Idealny model: Powinien dążyć do lewego górnego rogu (0,1), co oznacza wysoki TPR i niski FPR
- Linia losowego klasyfikatora: Linia 45 stopni od (0,0) do (1,1) reprezentuje losowe zgadywanie. Model powyżej tej linii jest lepszy niż losowe zgadywanie, a poniżej - gorszy

Wnioski:

1. **Las Losowy (Random Forest):**
   - **Wynik:** AUC ≈ **0.9999** (Linia fioletowa).
   - **Interpretacja:** Model jest niemal bezbłędny. Krzywa "przytula się" do lewego górnego rogu, co oznacza, że algorytm perfekcyjnie rozdziela klasy przy zachowaniu niemal zerowego poziomu fałszywych alarmów. Potwierdza to, że metoda zespołowa (wiele drzew) znacząco przewyższa pojedyncze drzewo.
2. **k-Najbliższych Sąsiadów (k-NN):**
   - **Wynik:** AUC ≈ **0.9980** (Linia zielona).
   - **Interpretacja:** Bardzo wysoka skuteczność. Model świetnie radzi sobie z mapowaniem podobieństw w danych sensorycznych, ustępując Lasowi Losowemu tylko w minimalnym stopniu.
3. **Drzewo Decyzyjne (Decision Tree):**
   - **Wynik:** AUC ≈ **0.9183** (Linia czerwona).

- **Interpretacja:** Najsłabszy wynik w zestawieniu. Charakterystyczny "schodkowy" kształt krzywej wynika z prostszej struktury decyzyjnej pojedynczego drzewa. Mimo przyzwoitego wyniku (>0.9), widać wyraźną różnicę klasy w porównaniu do metod bardziej zaawansowanych.

Podsumowanie wyników modeli klasyfikacyjnych

Metryka	Drzewo Decyzyjne	k-NN	Las Losowy (Random Forest)
Accuracy	97.58%	98.54%	**99.85%**
Recall	69.42%	89.57%	**98.92%**
Specificity	99.66%	99.21%	**99.92%**
AUC	0.9183	0.9980	**0.9999**

> **Wniosek:** Las Losowy okazał się bezkonkurencyjny we wszystkich kategoriach. Szczególną uwagę należy zwrócić na parametr **Recall (Czułość)**. Drzewo decyzyjne wykrywało obecność tylko w ok. 69% przypadków, podczas gdy Las Losowy robił to niemal bezbłędnie (99%).