Sztuczna inteligencja i inżynieria wiedzy

Lista 4

Konrad Kiełczyński 260409

1.Eksploracja danych

Zbiór dotyczy informacji na temat różnych typów szkła. Składa się z 214 rekordów i 11 kolumn.

```
print(df.shape)
```

```
(214, 10)
```

Dane statystyczne dotyczące zbioru danych

```
print(df.describe())
```

Python

	RI	Na	Mg	Al	Si	K	\
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	
mean	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	
std	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	
min	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	
25%	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	
50%	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	
75%	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	
max	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	

	Ca	Ba	Fe	class
count	214.000000	214.000000	214.000000	214.000000
mean	8.956963	0.175047	0.057009	2.780374
std	1.423153	0.497219	0.097439	2.103739
min	5.430000	0.000000	0.000000	1.000000
25%	8.240000	0.000000	0.000000	1.000000
50%	8.600000	0.000000	0.000000	2.000000
75%	9.172500	0.000000	0.100000	3.000000
max	16.190000	3.150000	0.510000	7.000000

Liczba poszczególnych danych dotyczących poszczególnych klas

```
print(df['class'].value_counts())
```

```
2    76
1    70
7    29
3    17
5    13
6     9
Name: class, dtype: int64
```

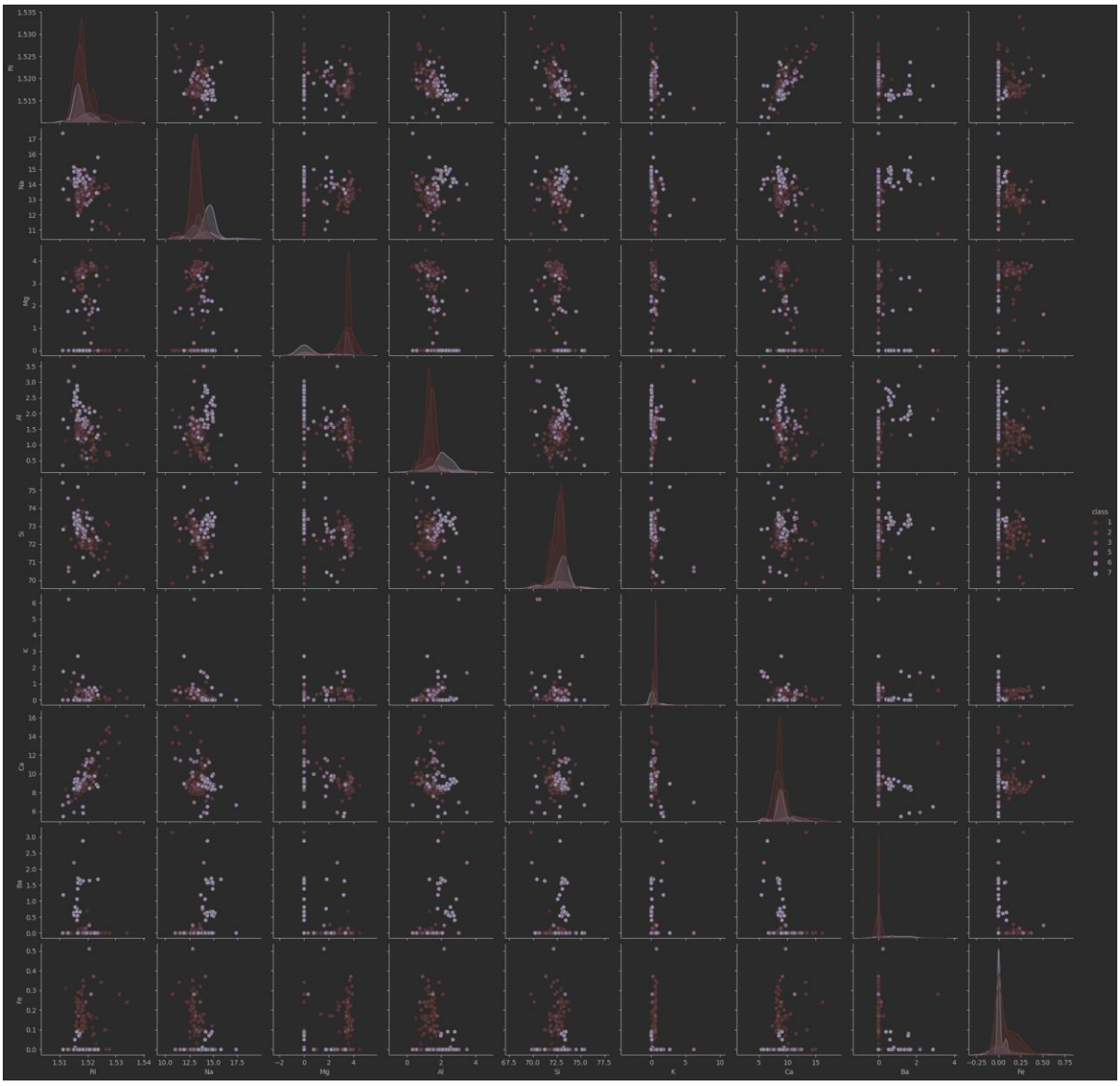
Na podstawie podanych danych można wysnuć następujące wnioski:

- Klasa 2 ma największą liczbę wystąpień (76).
- Klasa 1 jest drugą najczęściej występującą klasą (70).
- Klasy 7, 3, 5 i 6 mają mniejsze liczby wystąpień: odpowiednio 29, 17, 13 i 9.

Podane dane sugerują, że zbiór danych może być niezrównoważony, ponieważ nie wszystkie klasy mają podobną liczbę wystąpień. Klasy 2 i 1 są dominującymi klasami, podczas gdy klasy 7, 3, 5 i 6

mają znacznie mniejsze liczby wystąpień. Istnienie takiego niezrównoważenia może mieć wpływ na wyniki analizy i modelowania, zwłaszcza jeśli zależy nam na równomiernym uwzględnieniu wszystkich klas.

Wizualizacja danych na wykresie



Widać że duża część danych jest mało zrównoważona i wartości to po prostu ciąg 0 stąd wiele lin prostych prostopadłych.

2. Cross-validation

W procesie oceny wydajności modelu uczenia maszynowego istotne jest zastosowanie odpowiednich technik podziału dostępnych danych na zbiór treningowy i testowy. W przypadku, gdy zależy nam na uwzględnieniu różnorodności danych oraz dokładnej ocenie wydajności, warto skorzystać z walidacji krzyżowej (cross-validation).

Walidacja krzyżowa polega na podziale danych na k podzbiorów, zwanych foldami, gdzie k jest wybraną liczbą równą podziałom. Następnie, dla każdego podziału, model jest trenowany na k-1 foldach i testowany na pozostałym foldzie. Proces ten jest powtarzany k razy, tak aby każdy fold

został użyty jako zbiór testowy co najmniej raz. Ostateczna ocena wydajności modelu jest obliczana jako średnia wyników uzyskanych dla poszczególnych podziałów.

StratifiedKFold

StratifiedKFold to pewna odmiana **Kfold**. Ta metoda zapewnia, że każdy zestaw foldów, na który zostaje podzielony zbiór danych, zawiera w przybliżeniu taki sam procent próbek z każdej klasy docelowej, jak cały zbiór danych. Dzięki temu, nawet jeśli zbiór danych jest nierównomierny, podział jest bardziej reprezentatywny i bardziej wiarygodnie odzwierciedla rozkład klas w oryginalnym zbiorze.

```
cv = StratifiedKFold(n_splits=5, random_state=SEED, shuffle=True)
```

- **n_splits=5** oznacza, że zbiór danych zostanie podzielony na 5 foldów,
- **random_state=SEED** pozwala na ustalenie ziarna losowości, co zapewnia deterministyczny podział danych,
- **shuffle=True** powoduje, że dane są losowo mieszane przed podziałem, co jest przydatne, aby zmniejszyć wpływ sekwencji danych na wyniki walidacji.

3.Preprocessing

```
preprocessors = [
    ('None', None, {}),
    ('Normalization', StandardScaler(), {
        'preprocessing__with_mean': [True, False],
        'preprocessing__with_std': [True, False]
    }),
    ('Feature Selection', SelectKBest(score_func=f_classif), {
         'preprocessing__k': [1, 2, 3, 4, 5, 6, 7, 8, 9],
    }),
    ('PCA', PCA(), {
        'preprocessing__n_components': [1, 2, 3, 4, 5, 6, 7, 8, 9]
    })
]
```

1. **None (brak preprocesora):**

   Ten preprocesor oznacza, że nie ma żadnego przetwarzania danych przed ich przekazaniem do algorytmu uczenia maszynowego. Dane są używane w ich oryginalnej postaci.

2. **Normalization (standaryzacja):**

   Ten preprocesor stosuje standaryzację do danych, co polega na przekształceniu wartości cech tak, aby miały średnią równą 0 i odchylenie standardowe równa 1. Standaryzacja jest przydatna, gdy różne cechy mają różne skale, co może prowadzić do problemów w niektórych modelach.

3. **'Feature Selection' (wybór cech):**

   Ten preprocesor wykonuje selekcję cech, aby wybrać zestaw najbardziej istotnych cech dla modelu. W tym przypadku wykorzystuje funkcję **f_classif**, która oblicza statystykę F i p-value dla testu ANOVA dla każdej cechy. Argument **score_func** wskazuje na funkcję, która ma zostać użyta do oceny znaczenia cech. Parametr **k** oznacza liczbę cech, które mają zostać wybrane. W tym przypadku będą sprawdzane wartości od 1 do 9 (ilość cech).

4. **'PCA' (analiza składowych głównych):**

   PCA (Principal Component Analysis) to technika redukcji wymiarowości, która przekształca dane wejściowe w nowy zestaw nieskorelowanych zmiennych nazywanych składowymi głównymi. Parametr **n_components** wskazuje liczbę głównych składowych, które mają zostać zachowane. W tym przypadku będą sprawdzane wartości od 1 do 9 (ilość cech).

4. Klasyfikacja

```
classifiers = [
    ('Naive Bayes', GaussianNB(), {
        'classifier__var_smoothing': [1e-9, 1e-8, 1e-7]
    }),
    ('Decision Tree', DecisionTreeClassifier(random_state=SEED), {
        'classifier__max_depth': [None, 5, 10],
        'classifier__criterion': ['gini', 'entropy']
    }),
    ('Random Forest', RandomForestClassifier(random_state=SEED), {
        'classifier__n_estimators': [100, 200],
        'classifier__max_depth': [None, 5]
    })
]
```

1. **Naive Bayes (GaussianNB):**

   Naive Bayes to prosty i efektywny algorytm klasyfikacji, który bazuje na zastosowaniu twierdzenia Bayesa z założeniem o niezależności cech. Opiera się na twierdzeniu Bayesa oraz założeniu o niezależności cech. Algorytm zakłada, że wszystkie cechy są niezależne od siebie, co oznacza, że obecność danej cechy w klasie jest niezależna od obecności innych cech.

   - var_smoothing: reprezentuje on część największej wariancji ze wszystkich cech i jest dodawany do wariancji wszystkich cech w celu zapewnienia stabilności obliczeń.

2. **Decision Tree (DecisionTreeClassifier):**

   Drzewa decyzyjne są graficznym modelem używanym do podejmowania decyzji lub tworzenia prognoz. Klasyfikator Decision Tree dokonuje podziału danych na podstawie różnych cech, tworząc hierarchiczne drzewo decyzyjne. Każdy liść drzewa decyzyjnego reprezentuje konkretną klasę lub etykietę. Podział odbywa się poprzez wybór cechy i wartości progowej, które najlepiej separują próbki. Algorytm podejmuje decyzję o klasyfikacji na podstawie ścieżki prowadzącej od korzenia drzewa do odpowiedniego liścia

   - max_depth: Im większa wartość max_depth, tym bardziej skomplikowane i bardziej dopasowane do danych drzewo może być. Należy jednak uważać na overfitting, gdy wartość max_depth jest zbyt duża (None, 5, 10)
   - criterion: To parametr określający kryterium oceny jakości podziału w drzewie decyzyjnym. (gini, entropy)

3. **Random Forest (RandomForestClassifier):**

Random Forest to zespołowy algorytm uczenia maszynowego oparty na drzewach decyzyjnych. Wykorzystuje kombinację wielu drzew decyzyjnych w celu poprawy skuteczności klasyfikacji. Każde drzewo w Random Forest jest niezależnym klasyfikatorem, który jest trenowany na losowo wybranym podzbiorze danych ze zwracaniem.

- n_estimators: liczba drzew w lesie (100, 200)
- max_depth: maksymalna głębokość drzewa (None, 5)

ZALETY:

- Redukcja wariancji: Losowy wybór próbek i cech dla każdego drzewa oraz agregacja predykcji sprawiają, że Random Forest jest stabilniejszy i mniej podatny na przeuczenie niż pojedyncze drzewa decyzyjne.
- Ważność cech: Random Forest dostarcza informacji o ważności cech. Mierzy się wpływ każdej cechy na poprawność predykcji i tworzy ranking ważności cech, co pomaga zrozumieć, które cechy mają większy wpływ na klasyfikację.
- Efektywność obliczeniowa: Random Forest może równolegle budować drzewa decyzyjne, co przyspiesza proces uczenia w porównaniu do innych algorytmów, takich jak boosting.

4.Metryki

- **Accuracy (Dokładność):** Mierzy procentowy stosunek poprawnie sklasyfikowanych przypadków do całkowitej liczby przypadków. Jest to ogólna miara skuteczności klasyfikatora i informuje nas o tym, jak często model poprawnie przewiduje klasy.

- **Precision (Precyzja):** Mierzy zdolność klasyfikatora do identyfikacji poprawnych pozytywnych przypadków. Oznacza to, że mierzy, jak często klasyfikator poprawnie przewiduje, że dany przypadek jest pozytywny spośród wszystkich przypadków, które sklasyfikował jako pozytywne. Precyzja jest przydatna, gdy zależy nam na minimalizacji fałszywie pozytywnych wyników.

- **Recall (Czułość):** Mierzy zdolność klasyfikatora do wykrywania wszystkich prawdziwych pozytywnych przypadków. Oznacza to, jak często klasyfikator poprawnie przewiduje, że dany przypadek jest pozytywny spośród wszystkich rzeczywiście pozytywnych przypadków. Czułość jest przydatna, gdy zależy nam na minimalizacji fałszywie negatywnych wyników.

    - **F1 Score:** Jest to harmoniczna średnia precyzji i czułości. F1 Score łączy obie metryki w jedną wartość, co pozwala ocenić równocześnie zarówno precyzję, jak i czułość. Jest przydatny, gdy zależy nam na osiągnięciu równowagi między precyzją a czułością.

5.Główny algorytm

W ramach eksperymentu, wykonano iteracyjne przeszukiwanie siatki (GridSearchCV) w celu znalezienia optymalnych kombinacji preprocesorów i klasyfikatorów dla danego zadania klasyfikacji.

```
results = []

cv = StratifiedKFold(n_splits=5, random_state=SEED, shuffle=True)


for prep_name, preprocessor in preprocessors:
    for clf_name, classifier, params in classifiers:
        pipeline = create_pipeline(classifier, preprocessor)

        grid_search = GridSearchCV(pipeline, params, scoring='f1_micro',
cv=cv)
        grid_search.fit(X, y)
        best_estimator = grid_search.best_estimator_

        # Collect the results
        result = {
            'Preprocessor': prep_name,
            'Classifier': clf_name,
            'Best Parameters': grid_search.best_params_,
            'F1 Score': grid_search.best_score_
        }
        results.append(result)
```

1. Dla każdej kombinacji tworzony jest **pipeline** za pomocą funkcji **create_pipeline**, która łączy klasyfikator i preprocessor w jedną sekwencję przetwarzania.
2. Następnie dla danego **pipeline** tworzony jest obiekt **grid_search** klasy GridSearchCV. Parametry przekazywane do GridSearchCV to **pipeline** (model do optymalizacji), **params** (słownik z możliwymi kombinacjami hiperparametrów), **scoring='f1_micro'** (miara jakości, która jest optymalizowana w tym przypadku f1) oraz **cv=cv** (obiekt kroswalidacji, w tym przypadku StratifiedKFold).
3. Metoda **fit** jest wywoływana na obiekcie **grid_search**, co powoduje wykonanie procesu optymalizacji hiperparametrów. GridSearchCV przeszukuje siatkę parametrów i dla każdej kombinacji wykonuje walidację krzyżową na zbiorze danych.
4. Po zakończeniu optymalizacji dla danej kombinacji preprocessora i klasyfikatora, otrzymujemy **best_estimator** - model z optymalnymi hiperparametrami. Oraz **best_score_** który jest średnia wyniku walidacji krzyżowej dla najlepszego estymatora.

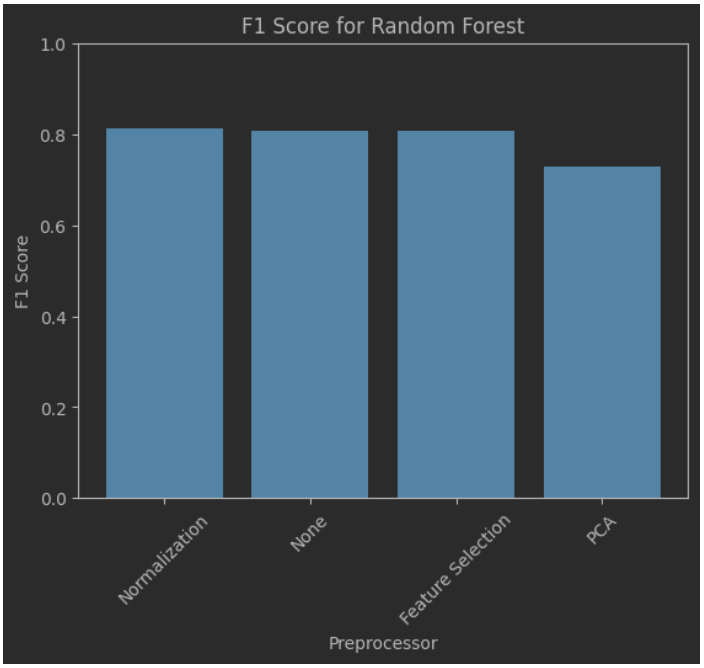
6.Analiza wyników

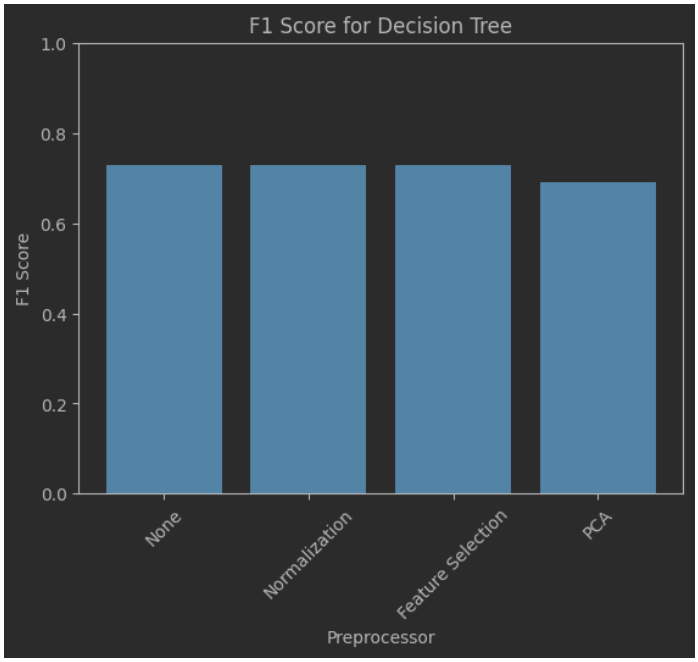
Najlepszą kombinacją okazała się

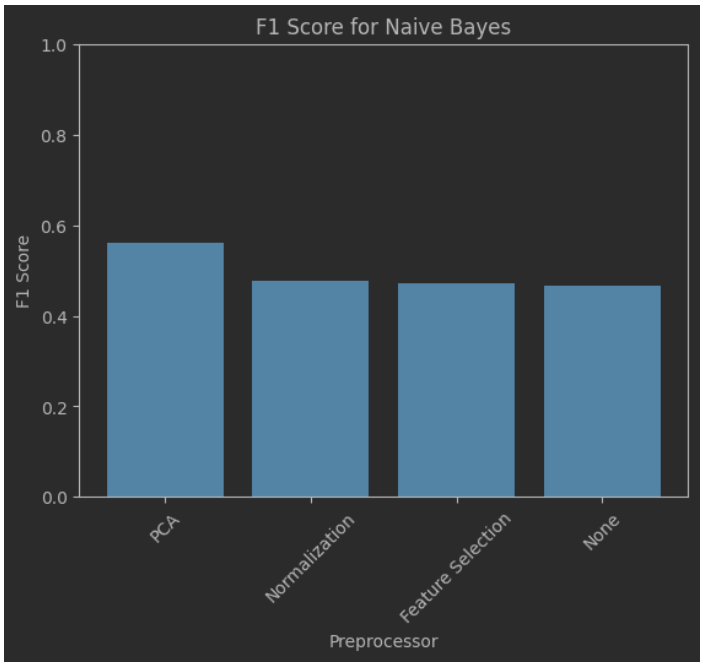

```
[
    {
        "Preprocessor": "Normalization",
        "Classifier": "Random Forest",
        "Best Parameters": {
            "classifier__max_depth": null,
            "classifier__n_estimators": 200,
            "preprocessing__with_mean": true,
            "preprocessing__with_std": false
        },
        "F1 Score": 0.8128460686600221
```

Porównanie różnych Klasyfikatorów i preprocesorów

Na wykresie pokazane są wyniki poszczególnych najlepszych estymatorów z najlepszymi hiperparametrami oraz porówanie wartości F1 score dla nich.







Z wykresów widać że najlepsze wyniki osiągał klasyfikator Lasu Losowego, potem troszkę odstające wyniki dał klasyfikator Drzewa decyzyjnego jednak zdecydowanie najgorszy okazał się natywny klasyfikator Bayesa.

W przypadku drzewa losowego preprocesing (normalizacja) dał małe polepszenie wyniku. Natomiast w przypadku drzewa decyzyjnego preprocesing dał takie same wyniki jak bez niego.

Ciekawym faktem jest to że PCA który pogarszał wyniki preprocesingu w innych klasyfikatorach znacznie polepszył wynik w natywnym Bayesie.

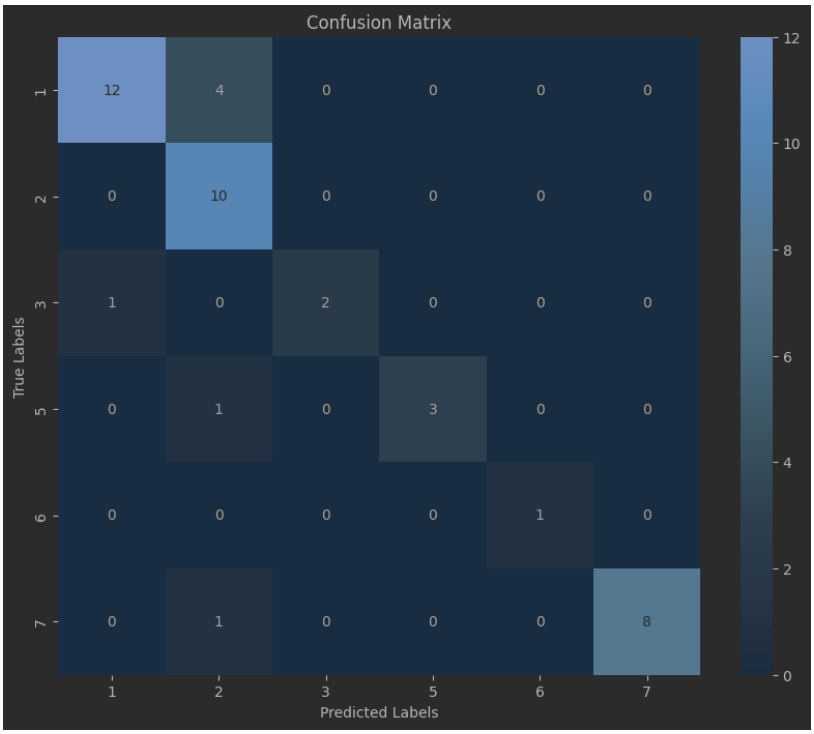
Drzewa decyzyjne i lasy losowe są technikami opartymi na regułach decyzyjnych, które mogą być wrażliwe na zmniejszenie wariancji danych wprowadzone przez PCA. Jeśli dane są bardzo zróżnicowane i mają skomplikowaną strukturę, to redukcja wymiarów dokonana przez PCA może usunąć niektóre ważne cechy, które były istotne dla tych modeli

Najlepszy wytrenowany model i metryki oraz macierz pomyłek

Z algorytmu wyszło że najlepszym modelem jest random forrest z normalizacja.

Wartości metryk dla tego pipeline:
'F1 Score': 0.8372093023255814,
'Accuracy': 0.8372093023255814,
'Recall': 0.8372093023255814,
'Precision': 0.8372093023255814

Confusion Matrix



Z macierzy można wyczytać że najtrudniej było sklasyfikować klasę nr 2 ma ona najwięcej pomyłek. Co ciekawe klasa ta była najczęściej mylona z klasą numer 1 lecz nie odwrotnie.

Solution:
https://github.com/Golden3x11/AI/tree/main/ml