Sprawozdanie - elementy inteligencji obliczeniowej (EIO)	
Prowadzący: *mgr inż. Anna Labijak-Kowalska*	Grupa: ***L9***
Temat Ćwiczeń: *Klasyfikator **KNN***	
Autorzy: *Daniel Zdancewicz [145317]*	

1 Wstęp - Cel Ćwiczenia

Celem ćwiczenia było wykorzystanie klasyfikatora KNN do oceny czerwonego wina jako:

- "good": Dobre – Wino, które jest pitne.
- "medium": Takiese – Wino, które jest takiese.
- "poor": Słabe – Wino, które jest niepitne.

Należało tego dokonać na podstawie 1600 dostępnych rekordów z przygotowanego wcześniej zbioru danych winequality-red.csv. Wpierw zbiór danych potrzeba było znormalizować i ustandaryzować, następnie podzielić na zbiór treningowy i testowy. Na podstawie tak przygotowanych zbiorów eksperymentalnie określić najlepszą liczbę k dla algorytmu, a następnie dokonać klasyfikacji przy wykorzystaniu obranej liczby k.

2 Odczyt i preprocessing

Normalizacja odbyła się poprzez wykorzystanie metody fit_transform dostępnej w sklearn.preprocessing.MinMaxScaler wykonany na wartościach numerycznych win. Zbiór danych został spróbkowany by była zwiększona w nim entropia i wyniki nie były zależne od kolejności danych.

```
import numpy as np
import sklearn.model_selection as skm
import sklearn.preprocessing as skp
import sklearn.neighbors as skn
import pandas as pd

def read_data(path: str) -> pd.DataFrame:
  return pd.read_csv(path)

def preprocess_df(df: pd.DataFrame) -> pd.DataFrame:
  df = df.sample(frac=1).reset_index(drop=True)
  classes = df['quality']
```

```
  df = df.drop('quality', axis=1)

  normalized_df = skp.MinMaxScaler().fit_transform(df)
  return normalized_df, classes.values
```

3 Podział zbioru treningowego i testowego

Podział na zbiór treningowy i testowy został wykonany poprzez metodę train_test_skip dostępną w sklearn.model_selection.

```
def split_df(data_X, data_Y, test_percent=0.2, random_state=5) -> (
    (pd.DataFrame, list[str]), (pd.DataFrame, list[str])):
  (X_train, X_test, Y_train, Y_test) = skm.train_test_split(
    data_X, data_Y, test_size=test_percent, random_state=random_state
  )
  return ((X_train, Y_train), (X_test, Y_test))
```

4 Eksperymentalne poszukiwanie najlepszego K

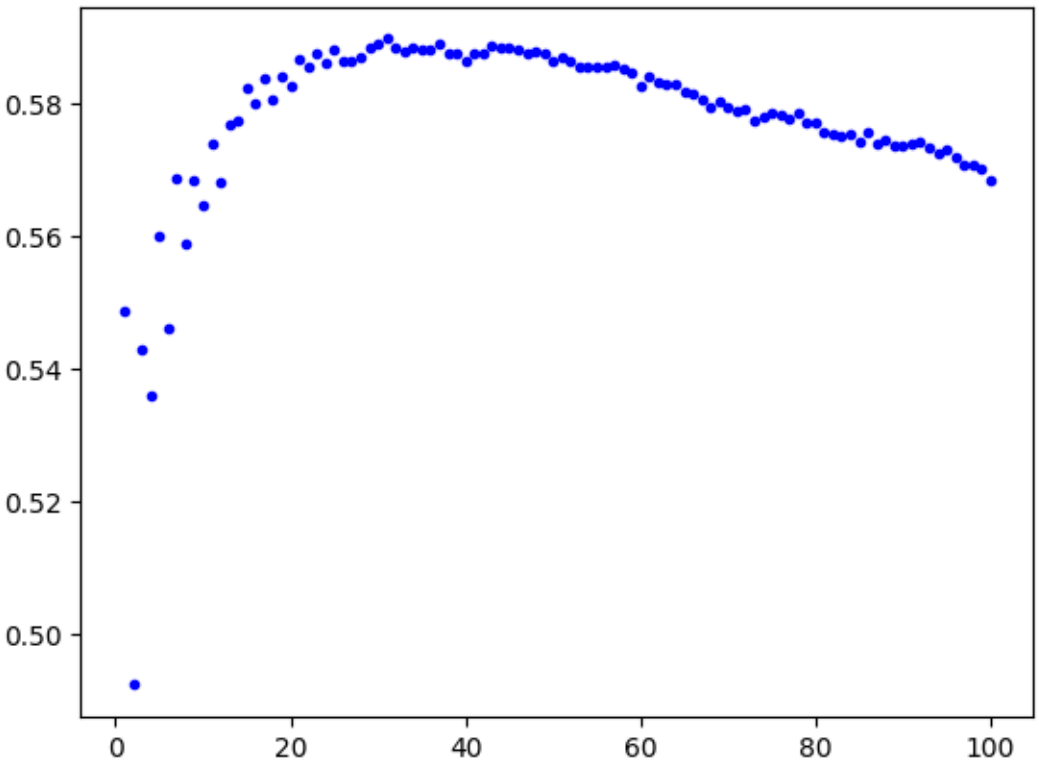
Najlepsze K zostało odnalezione poprzez wielokrotne (100 w wykresie) trenowanie i testowanie za pomocą walidacji krzyżowej o cv=10.

Kod implementujący przeszukiwanie:

```
def calculate_cross_acc(k: int):
  classifier = skn.KNeighborsClassifier(n_neighbors=k)
  classifier.fit(*train_split)

  return np.mean(skm.cross_val_score(classifier, *test_split))


wine_df = read_data("resources/winequality-red.csv")
items = []
for _ in range(100):
  (train_split, test_split) = split_df(*preprocess_df(wine_df))

  accuracies_pairs = tuple(
    enumerate(map(calculate_cross_acc, range(1, 100 + 1)), start=1)
  )
  items.append(accuracies_pairs)
averages = tuple(map(lambda x: np.average(x, axis=0), zip(*items)))
```

4.1 Tworzenie wykresu na podstawie średnich wyników wielokrotnego przeszukiwania

```
import matplotlib.pyplot as plt

plt.plot(*zip(*averages), 'bo', markersize=3)

(best_k, score) = max(averages, key=lambda x: x[1])
print(f"Best score for k={int(best_k)} with {score * 100:.2f}% accuracy")
```



Rysunek 1: Wykres obrazujący celność walidacji krzyżowej wobec k na zbiorze

5 Wyniki eksperymentu

Jak można zauważyć najlepsze K jest wokół górki, czyli w okolicach 30. Zazwyczaj wychodziło 31 o celności 58%.