Maciej Zagórski
Informatyka, III rok

RAPORT Z PROJEKTU
Sprawdzenie działania wybranych klasyfikatorów

1. Przedmiot projektu

Przedmiotem projektu było sprawdzenie, jak działają następujące, poznane klasyfikatory (metody, algorytmy uczenia maszynowego (nadzorowanego)):

- naiwny klasyfikator Bayesa (*Naive Bayes*),
- klasyfikator k-najbliższych sąsiadów (*k-Nearest Neighbors*, k-NN) – z różnym parametrem k:
  - k = 3
  - k = 5
  - k = 11
- algorytm wykorzystujący drzewa decyzyjne (*Decision tree*).

Sprawdzenia działania powyższych klasyfikatorów dokonano w oparciu o przykładowy zbiór danych „diabetes.csv", zawierający informacje o kobietach indiańskiego pochodzenia z USA (ich parametry medyczne), które zachorowały lub nie zachorowały na cukrzycę. Zastosowanie klasyfikatora w takim wypadku służy diagnozowaniu choroby na podstawie znanych danych medycznych.

2. Realizacja projektu

2.1. Analiza zbioru danych

Zbiór danych „diabetes.csv", jak wskazano, zawiera parametry medyczne grupy kobiet – 768 – takie jak: liczba ciąż (kolumna *Pregnancies*), wartość glukozy (*Glucose*), wartość ciśnienia krwi (*BloodPressure*), grubość skóry (*SkinThickness*), wartość insuliny (*Insulin*), wskaźnik BMI, wskaźnik występowania cukrzycy w rodzinie (*DiabetesPedigreeFunction*) oraz wiek (*Age*).

W zbiorze zawarto też informację o tym, czy dana osoba zachorowała na cukrzycę, czy nie – w kolumnie *Outcome*, przy pomocy wartości „0" lub „1". Na potrzeby realizacji niniejszego projektu przyjęto, że w kolumnie *Outcome* wartość „1" („prawda", *true*) oznacza, że dana osoba zachorowała na cukrzycę (odpowiednio: „0", jako „fałsz", *false* – nie zachorowała).

Poza powyższymi nazwami kolumn oraz wartościami w zbiorze danych nie zawarto informacji np. o jednostkach, w jakich wyrażane są poszczególne wartości, bądź też w jaki sposób zostały one wyliczone (np. w przypadku *DiabetesPedigreeFunction*).

Przedmiotowy zbiór danych nie zawiera danych porządkowych (w postaci np. kolumny zawierającej liczby porządkowe poszczególnych rekordów), które mogłyby nieadekwatnie wpływać („wypaczać") efekty zastosowania poszczególnych klasyfikatorów. Wobec tego przygotowanie zbioru danych do jego dalszego wykorzystania z tej perspektywy nie było konieczne.

Należy jednak zwrócić uwagę, że nie wszystkie dane zawarte w zbiorze są prawidłowe (pełne). Przykładowo: w kilku przypadkach wartość ciśnienia krwi została wskazana jako „0”, co mogłoby wskazywać, że badane osoby, w momencie wykonywania badań, np. nie żyły; w ponad 200 przypadkach wartość grubości skóry to „0”, co by mogło z kolei oznaczać, że badania nie wykonano.

2.2. Podział zbioru danych

W celu realizacji projektu w pierwszej kolejności zaimportowano do programu zbiór danych „diabetes.csv”, wykorzystując w tym celu bibliotekę *pandas*, przeznaczoną do analizy i manipulowania danymi. Tak zaimportowany zbiór podzielono następnie na dwa odrębne podzbiory:

- podzbiór „x”, zawierający dane medyczne (kolumny) poszczególnych osób, ale bez informacji o tym, czy osoba ta zachorowała, czy nie zachorowała na cukrzycę (bez kolumny *Outcome*),
- podzbiór „y”, zawierający informację wynikową o tym, czy dana osoba (o określonych parametrach medycznych) zachorowała na cukrzycę, czy też nie (tylko kolumna *Outcome*).

```
data_diabetes = pd.read_csv('../data/report_1_classifiers_data_diabetes.csv')

y = data_diabetes['Outcome']
x = data_diabetes.drop('Outcome', axis=1)
```

2.3. Normalizacja danych

Na tym etapie realizacji projektu dokonano również normalizacji danych – chcąc przetestować, czy rezultaty zastosowania poszczególnych klasyfikatorów będą się różnić w zależności od tego, czy będą one działać w oparciu o dane znormalizowane, czy też nie. W tym celu dokonano normalizacji danych zawartych w poszczególnych kolumnach podzbioru „x” – w obrębie tych kolumn – przy zastosowaniu następującej funkcji normalizującej *(normalize).*

```
def normalize(data, column):
    return (data[column] - data[column].min()) /
           (data[column].max() - data[column].min())
# [...]
x_norm = x.copy()

for column in x_norm:
    x_norm[column] = normalize(x_norm, column)
```

Ponieważ informacje zawarte w podzbiorze „y” to informacje wynikowe, o wartościach „0” lub „1” (odpowiednio: osoba zachorowała lub nie zachorowała na cukrzycę), wykonanie takiej normalizacji w odniesieniu do tych danych nie było konieczne.

2.4. Podział danych na dane treningowe i testowe

Następnie dokonano losowego podziału danych na podzbiory danych treningowych (67%) oraz testowych (33%). W tym celu wykorzystano funkcję *train_test_split* z modułu *sklearn.model_selection* (z biblioteki *scikit-learn*). Taki podział wykonano osobno dla danych „surowych”, nieznormalizowanych, oraz dla danych znormalizowanych (zmienne z infiksem „*_norm_*”).

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33,
                                                    random_state=0, stratify=y)
x_norm_train, x_norm_test, y_norm_train, y_norm_test = train_test_split(x_norm, y,
                                                                        test_size=0.33,
                                                                        random_state=0,
                                                                        stratify=y)
```

Zastosowano przy tym parametr *stratify=y,* dzięki któremu podzbiory zwracane przez funkcję *train_test_split* zachowały takie same proporcje etykiet klas, jak wejściowy zbiór danych.

2.5. Zastosowanie klasyfikatorów

W następnym kroku do tak przygotowanych podzbiorów danych zastosowano klasyfikatory wskazane w p. 1 wyżej – każdy osobno dla danych nieznormalizowanych oraz dla danych znormalizowanych. W efekcie uruchomienia każdego z nich wyświetlona została jego procentowa dokładność oraz macierz błędu. W tym celu wykorzystano, odpowiednio, metodę *score* w odniesieniu do każdego z klasyfikatorów (z odpowiednich modułów biblioteki *scikit-learn*) oraz funkcję *confusion_matrix* z modułu *metrics* tej biblioteki (na osi x wskazywane są rzeczywiste wartości, natomiast na osi y – wartości przewidywane przez klasyfikator).

- Naiwny klasyfikator Bayesa (*Naive Bayes*):

```
print("\n* * * Naive Bayes * * *\n")

gnb = GaussianNB()

gnb.fit(x_train, y_train)
y_pred = gnb.predict(x_test)

print("Score: %f" % (gnb.score(x_test, y_test)))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred), "\n")

print("Normalized:")

gnb.fit(x_norm_train, y_norm_train)
y_norm_pred = gnb.predict(x_norm_test)

print("Score: %f" % ((y_norm_test == y_norm_pred).sum() / x_norm_test.shape[0]))
print("Confusion matrix:\n", confusion_matrix(y_norm_test, y_norm_pred), "\n")
```

```
* * * Naive Bayes * * *

Score: 0.767717
Confusion matrix:
 [[141  24]
 [ 35  54]]

Normalized:
Score: 0.767717
Confusion matrix:
 [[141  24]
 [ 35  54]]
```

- Klasyfikator k-najbliższych sąsiadów (*k-Nearest Neighbors,* k-NN) – z parametrem k o wartości, odpowiednio, 3, 5 i 11, oraz przy zastosowaniu odległości euklidesowej (*p=2*) i metryki Minkowskiego (*metric='minkowski'*):

```
print("* * * k-NN * * *\n")

k_list = [3, 5, 11]
for k in k_list:
    print("k-%d neighbors:\n" % (k))


    knn = KNeighborsClassifier(n_neighbors=k, p=2, metric='minkowski')

    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)

    print("Score: %f" % (knn.score(x_test, y_test)))
    print("Confusion matrix:\n", confusion_matrix(y_test, y_pred), "\n")

    print("Normalized:")

    knn.fit(x_norm_train, y_norm_train)
    y_norm_pred = knn.predict(x_norm_test)

    print("Score: %f" % (knn.score(x_norm_test, y_norm_test)))
    print("Confusion matrix:\n", confusion_matrix(y_norm_test, y_norm_pred), "\n")
```

```
* * * k-NN * * *

k-3 neighbors:

Score: 0.692913
Confusion matrix:
 [[129  36]
 [ 42  47]]

Normalized:
Score: 0.755906
Confusion matrix:
 [[138  27]
 [ 35  54]]

k-5 neighbors:

Score: 0.732283
Confusion matrix:
 [[137  28]
 [ 40  49]]

Normalized:
Score: 0.748031
Confusion matrix:
 [[142  23]
 [ 41  48]]

k-11 neighbors:

Score: 0.736220
Confusion matrix:
 [[141  24]
 [ 43  46]]

Normalized:
Score: 0.748031
Confusion matrix:
 [[141  24]
 [ 40  49]]
```

- Algorytm wykorzystujący drzewa decyzyjne (*Decision tree*), z wykorzystaniem współczynnika Giniego jako kryterium rozgałęzień *(criterion='gini')* oraz określeniem maksymalnej głębokości (wysokości) drzewa jako 4 *(max_depth=4)*:

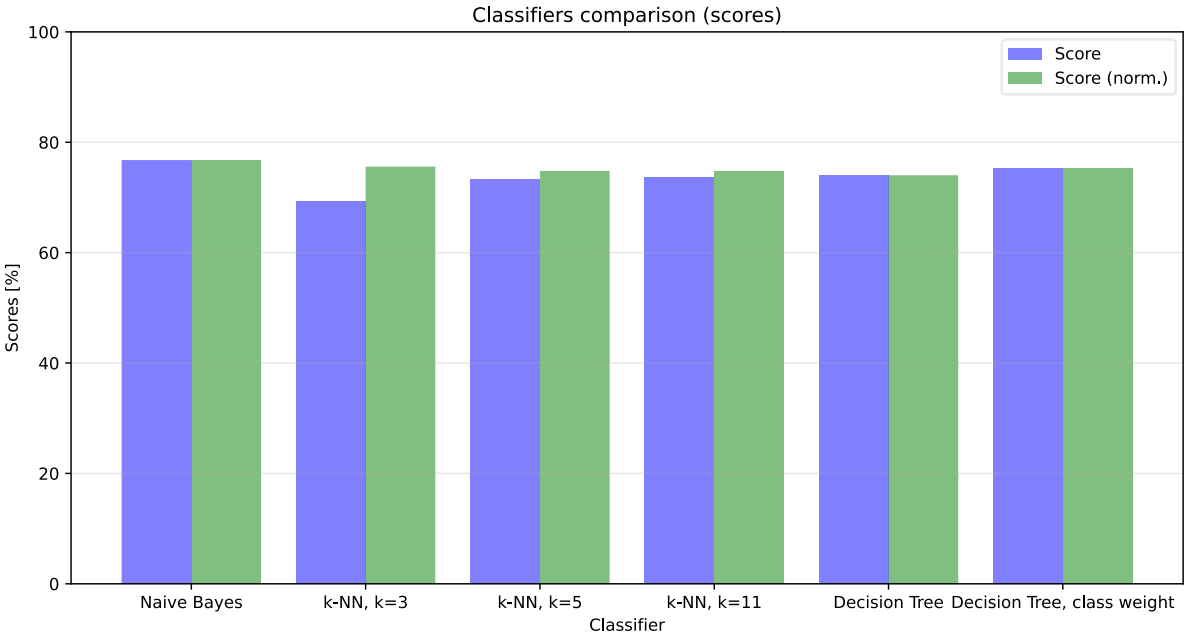
```
print("* * * Decision Tree * * *\n")

tree = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=1)

tree.fit(x_train, y_train)
y_pred = tree.predict(x_test)

print("Score: %f" % (tree.score(x_test, y_test)))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred), "\n")

print("Normalized:")

tree.fit(x_norm_train, y_norm_train)
y_norm_pred = tree.predict(x_norm_test)

print("Score: %f" % (tree.score(x_norm_test, y_norm_test)))
print("Confusion matrix:\n", confusion_matrix(y_norm_test, y_norm_pred), "\n")
```

```
* * * Decision Tree * * *

Score: 0.740157
Confusion matrix:
 [[128  37]
 [ 29  60]]

Normalized:
Score: 0.740157
Confusion matrix:
 [[128  37]
 [ 29  60]]
```

3. Prezentacja wyników

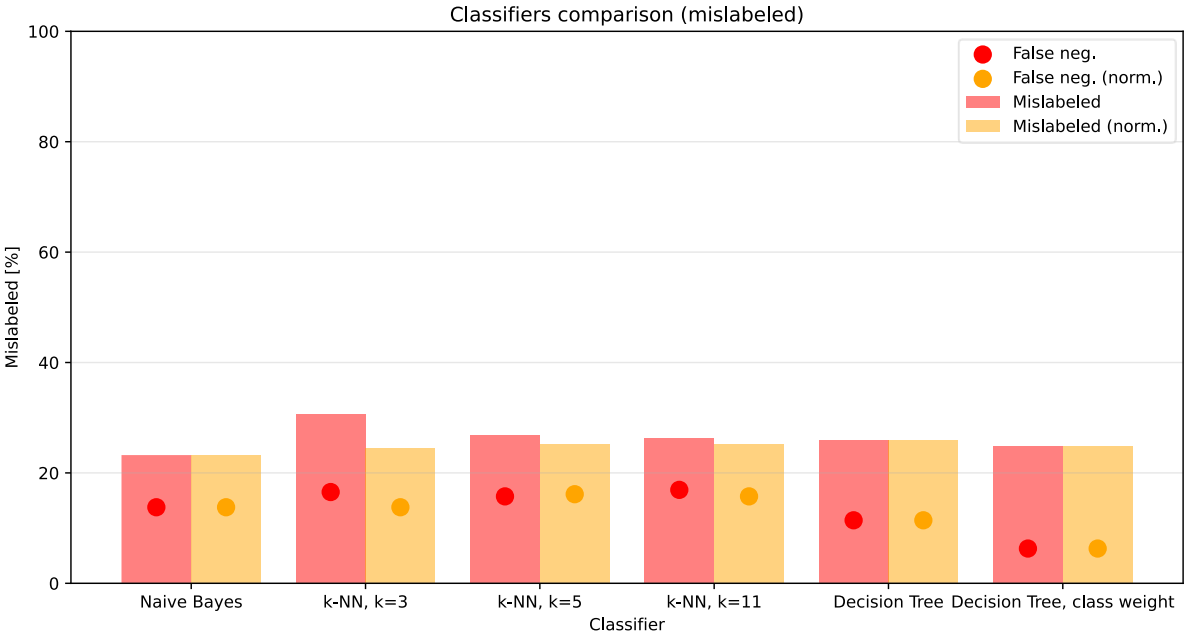
Uzyskane powyżej wyniki przedstawiono następnie w formie tabeli dla informacji o dokładności procentowej klasyfikatora oraz o zwracanych przez niego wynikach *false negative* oraz w formie wykresu słupkowego prezentującego dane o dokładności procentowej klasyfikatorów.

	Classifier	***Score***	***Score (norm.)***	***False neg.***	***False neg. (norm.)***
1.	*Naive Bayes*	0.767717	0.767717	35.0	35.0
2.	*k-NN, k=3*	0.692913	0.755906	42.0	35.0
3.	*k-NN, k=5*	0.732283	0.748031	40.0	41.0
4.	*k-NN, k=11*	0.736220	0.748031	43.0	40.0
5.	*Decision Tree,*	0.740157	0.740157	29.0	29.0
6.	*Dec. Tree, class weight*	0.751969	0.751969	16.0	16.0



Wartości w kategoriach oznaczonych *(norm.)*, np. *Score (norm.)* przedstawiono wyniki uzyskane przy wykorzystaniu danych znormalizowanych. W tabeli i na wykresie przedstawiono również wyniki zastosowania klasyfikatora *Decision tree* z parametrem *class weight* – uwagi w tym zakresie przedstawiono w p. 4 niżej.

Dodatkowo poniżej przedstawiono wykres porównujący niedokładności procentowe klasyfikatorów („odwrócenie" pierwszego wykresu), wraz z zaznaczonymi na nim wynikami *false negative* zwracanymi przez każdy z klasyfikatorów (wyrażonymi jako procent łącznych wyników danego klasyfikatora).



4. Komentarze i wnioski

W oparciu o uzyskane wyniki można wskazać, że:

- Najwyższą dokładnością (sięgającą blisko 77%) wykazał się klasyfikator *Naive Bayes,* najniższą zaś – klasyfikator k-NN z parametrem k równym 3, działający na danych nieznormalizowanych (ok. 69%).
- Usuwanie ze zbioru danych rekordów (przypadków), w odniesieniu do których nie zebrano pełnych danych medycznych (np. wspomniany w p. 2.1 wyżej brak informacji o ciśnieniu krwi) powodował, że niektóre klasyfikatory uzyskiwały lepsze wyniki – w zależności od tego, które dane były usuwane (jakie przyjęto kryterium usuwania danych; poniżej przedstawiono przykład usunięcia rekordów, dla których wartość ciśnienia krwi określono jako 0).

```
data_diabetes.drop(data_diabetes[(data_diabetes.BloodPressure == 0)].index,
                   inplace=True)
```

  Przy np. usunięcie rekordów, w których grubość skóry została określona jako 0, dokładności klasyfikatora Naive Bayes wzrosła do ponad 79%; natomiast usunięcie rekordów, w których wartość ciśnienia krwi wynosiła 0 spowodowało, że klasyfikator *Decision Tree*, przy określeniu maksymalnej wysokości drzewa jako 2 (*max_depth=2*), uzyskał dokładność powyżej 77%. Może to oznaczać, że odpowiednie przygotowanie danych, w połączeniu z wiedzą ekspercką i założeniem wyboru klasyfikatora (jego parametrów), **wpływa na dokładność (skuteczność) stosowanego rozwiązania.**

- Wyniki zastosowania klasyfikatora k-NN z różnymi wartościami parametru k przy wykorzystaniu danych nieznormalizowanych **były gorsze od jego zastosowań wykorzystujących dane znormalizowane** (w skrajnym przypadku k = 3 była to różnica w dokładności między ok. 69% a ok. 76%). Powyższe wyniknęło z tego, że zastosowany w ramach projektu klasyfikator k-NN wykorzystywał metrykę euklidesową (metrykę Minkowskiego będącą uogólnieniem metryki euklidesowej). Stosowanie tej metryki może wymagać normalizacji danych, tak aby każda cecha została odpowiednio wyskalowana (do odległości, w oparciu o które klasyfikator dokonuje wyboru).
- Jak wskazano w p. 1 wyżej, w sytuacji będącej przedmiotem projektu zastosowanie klasyfikatora służy diagnozowaniu choroby na podstawie znanych danych medycznych. W takim praktycznym kontekście dużo „gorszym" błędnym wynikiem jest uznanie przez algorytm, że określona osoba jest zdrowa (wartość „0" w kolumnie *Outcome*), podczas gdy jest ona w rzeczywistości chora (*Outcome* o wartość „1") – przypadek *false negative*. Z perspektywy społeczno-zdrowotnej lepiej (błędnie) uznać, że osoba zdrowa jest chora – przypadek *false positive* (który będzie mógł zostać zweryfikowany w drodze dalszych badań) – niż odwrotnie (odsyłając taką osobę, bez badań i leków, „do domu").
- W przypadku klasyfikatorów *Naive Bayes* oraz k-NN liczba przypadków *false negative* była większa od przypadków *false positive*. Odmienny rezultat dało zastosowanie klasyfikatora *Decision tree* – w jego wypadku liczba przypadków *false negative* (29) była mniejsza od przypadków *false positive* (37). Klasyfikator ten **pozwala też na zastosowanie parametru *class weight* – przyporządkowanie odpowiednich „wag" klasom, do których klasyfikowane są wyniki.** Ponieważ informację o chorobie (potencjalnym byciu chorym) uznajemy za ważniejszą niż o byciu zdrowym, klasie „1" w kolumnie *Outcome* należy przypisać odpowiednio większą wagę. Poniżej zaprezentowano zastosowanie przedmiotowego klasyfikatora przy użyciu parametru *class weight,* przypisującego wartość klasie „1" równą 1, natomiast klasie „0" – równą 0,25 (*class_weight={0:0.25, 1:1}*).

```
print("Using class weight:\n")

tree = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=1,
                              class_weight={0:0.25, 1:1})

tree.fit(x_train, y_train)
y_pred = tree.predict(x_test)

print("Score: %f" % (tree.score(x_test, y_test)))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred), "\n")

print("Normalized:")

tree.fit(x_norm_train, y_norm_train)
y_norm_pred = tree.predict(x_norm_test)

print("Score: %f" % (tree.score(x_norm_test, y_norm_test)))
print("Confusion matrix:\n", confusion_matrix(y_norm_test, y_norm_pred), "\n")
```

```
Using class weight:

Score: 0.751969
Confusion matrix:
 [[118  47]
 [ 16  73]]

Normalized:
Score: 0.751969
Confusion matrix:
 [[118  47]
 [ 16  73]]
```

Z powyższego wynika, że zastosowanie klasyfikatora z parametrem *class weight* dało niewiele gorszą dokładność niż w przypadku (najdokładniejszego) klasyfikatora *Naive Bayes* (ok. 75% vs. ok. 77%) – **przy jednocześnie ponad dwukrotnie niższej liczbie przypadków *false negative*** (16 vs. 35).

- Klasyfikator *Decision tree* pozwala też na ustalenie parametru maksymalnej „głębokości" (wysokości) drzewa decyzyjnego. Zaproponowaną w realizacji projektu wartość 4 (*max_depth=4*) określono na zasadzie prób i błędów – przy wartościach niższych lub wyższych klasyfikator zwracał gorsze wyniki (charakteryzował się gorszą dokładnością). Niższą dokładność przy większej wartości tego parametru można wytłumaczyć tym, że im większe drzewo, tym decyzje na jego kolejnych poziomach stają się coraz bardziej złożone, co może z kolei prowadzić do przetrenowania modelu.

5. Źródła

[b.a.], *Kaggle: Your Home for Data Science*, https://www.kaggle.com/, [dostęp: 28.04.2023 r.]

[b.a.], *Matplotlib – Visualization with Python*, https://matplotlib.org/, [dostęp: 28.04.2023 r.]

[b.a.], *pandas - Python Data Analysis Library*, https://pandas.pydata.org/, [dostęp: 28.04.2023 r.]

[b.a.], *scikit-learn: machine learning in Python – scikit-learn 1.2.2 documentation*, https://scikit-learn.org/stable/, [dostęp: 28.04.2023 r.]

S. Raschka, V. Mirjalili, *Python. Machine learning i deep learning. Biblioteki scikit-learn i TensorFlow 2*, wyd. 3, Gliwice 2021.