

Systemy sztucznej inteligencji

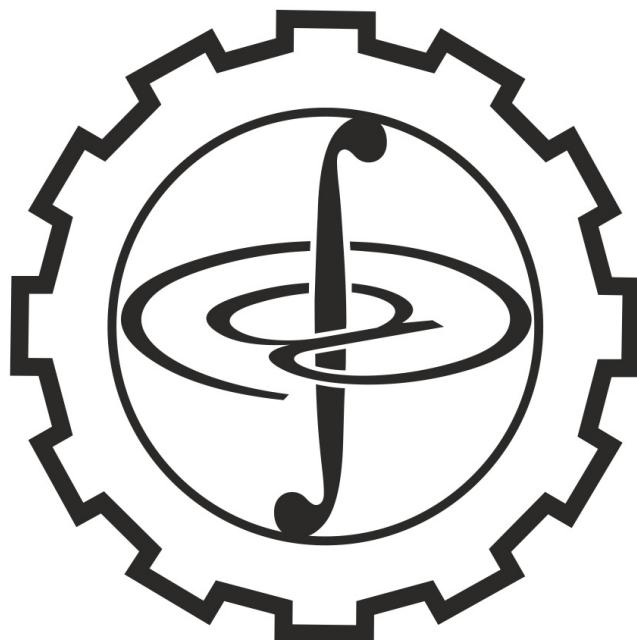
dokumentacja projektu „*Przewidywanie cen telefonów*”

Miłosz Flasz, gr. 4

Jakub Nocoń, gr. 1

Artur Łukaszek, gr. 2

Informatyka (stacjonarnie), Semestr IV



Politechnika Śląska
Wydział Matematyki Stosowanej
1 czerwca 2022

1 Część I

1.1 Opis projektu

Tematem naszego projektu jest przewidywanie cen telefonów komórkowych na podstawie ich różnych parametrów. Celem jest przygotowanie programu, który będzie dopasowywał cenę aparatu do zadanych atrybutów uwzględnionych w przyjętej przez nas bazie danych.

Najpierw przeprowadziliśmy analizę danych, której rezultaty znajdują się w adekwatnie zatytułowanym podrozdziale podrozdziale. Następnie w naszym projekcie wykorzystaliśmy algorytmy K najbliższych sąsiadów oraz drzewa decyzyjnego. Na końcu zamieściliśmy wnioski odnośnie wyników działania obu algorytmów oraz konkluzję, który z nich jest bardziej stosowny do przyjętej przez nas bazy danych.

1.2 Baza danych

W swoim projekcie wykorzystujemy bazę danych, w której zawarto cechy charakteryzujące telefony komórkowe. W bazie uwzględniono 21 różnych parametrów: wagę telefonu, liczbę rdzeni procesora, szerokość i wysokość telefonu, ilość pamięci RAM, dostęp do ekranu dotykowego, dostęp do WiFi, podwójne wejście karty SIM, czy czas działania baterii, pojemność baterii, dostęp do technologii Bluetooth, szybkość mikroprocesorów, ostrość przedniego i tylniego obiektywu, ilość pamięci wewnętrznej. Dane usystematyzowano za pomocą numeru identyfikacyjnego (ID).

1.3 Dodatkowe informacje

Baza danych liczy ok. 2000 rekordów, ale w obliczeniach ograniczyliśmy się do 400 rekordów. Głównym parametrem, do którego się odnosiliśmy była cena, którą opisują cyfry z zakresu 0-3:

- 0 - cena niska
- 1 - cena średnia
- 2 - cena wysoka
- 3 - cena bardzo wysoka

2 Część II

2.1 Opis działania

Teoria wykorzystanych algorytmów.

Algorytm **k najbliższych sąsiadów** (**k-nn** z ang. k nearest neighbours) - jeden z algorytmów regresji nieparametrycznej używanych w statystyce do prognozowania wartości pewnej zmiennej losowej. Używany również do klasyfikacji. Polega na:

1. porównaniu wartości zmiennych objaśniających dla obserwacji C z wartościami tych zmiennych dla każdej obserwacji w zbiorze uczącym;
2. wyborze k (ustalonej z góry liczby) najbliższych do C obserwacji ze zbioru uczącego
Definicja „najbliższych obserwacji” sprowadza się do minimalizacji pewnej metryki, mierzącej odległość pomiędzy wektorami zmiennych objaśniających dwóch obserwacji (np. metryki euklidesowej). Można również zamiast średniej arytmetycznej stosować np. medianę.
3. średnieniu wartości zmiennej objaśnianej dla wybranych obserwacji, w wyniku czego uzyskujemy prognozę.

Algorytm k najbliższych sąsiadów jest użyteczny szczególnie wtedy, gdy zależność między zmiennymi objaśniającymi a objaśnianymi jest złożona lub nietypowa (np. niemonotoniczna), czyli trudna do modelowania w klasyczny sposób. W przypadku, gdy zależność ta jest łatwa do interpretacji (np. liniowa), a zbiór nie zawiera obserwacji odstających, metody klasyczne (np. regresja liniowa) dadzą zwykle dokładniejsze wyniki.

Drzewo decyzyjne - algorytm stosowany w uczeniu maszynowym do pozyskiwania wiedzy na podstawie przykładów. Zadaniem drzew decyzyjnych może być zarówno stworzenie planu, jak i rozwiązanie problemu decyzyjnego. Metoda drzew decyzyjnych jest szczególnie przydatna w problemach decyzyjnych z licznymi, rozgałęziającymi się wariantami oraz w przypadku podejmowania decyzji w warunkach ryzyka.

Drzewa decyzyjne w uczeniu maszynowym służą do wyodrębniania wiedzy z zestawu przykładów. Zakładamy, że posiadamy zestaw przykładów: obiektów opisanych przy pomocy atrybutów, którym przyporządkowujemy jakąś decyzję. Drzewo odzwierciedla, w jaki sposób na podstawie atrybutów były podejmowane decyzje klasyfikujące. Algorytm działa rekurencyjnie dla każdego węzła drzewa. Musimy podjąć decyzje, czy węzeł będzie:

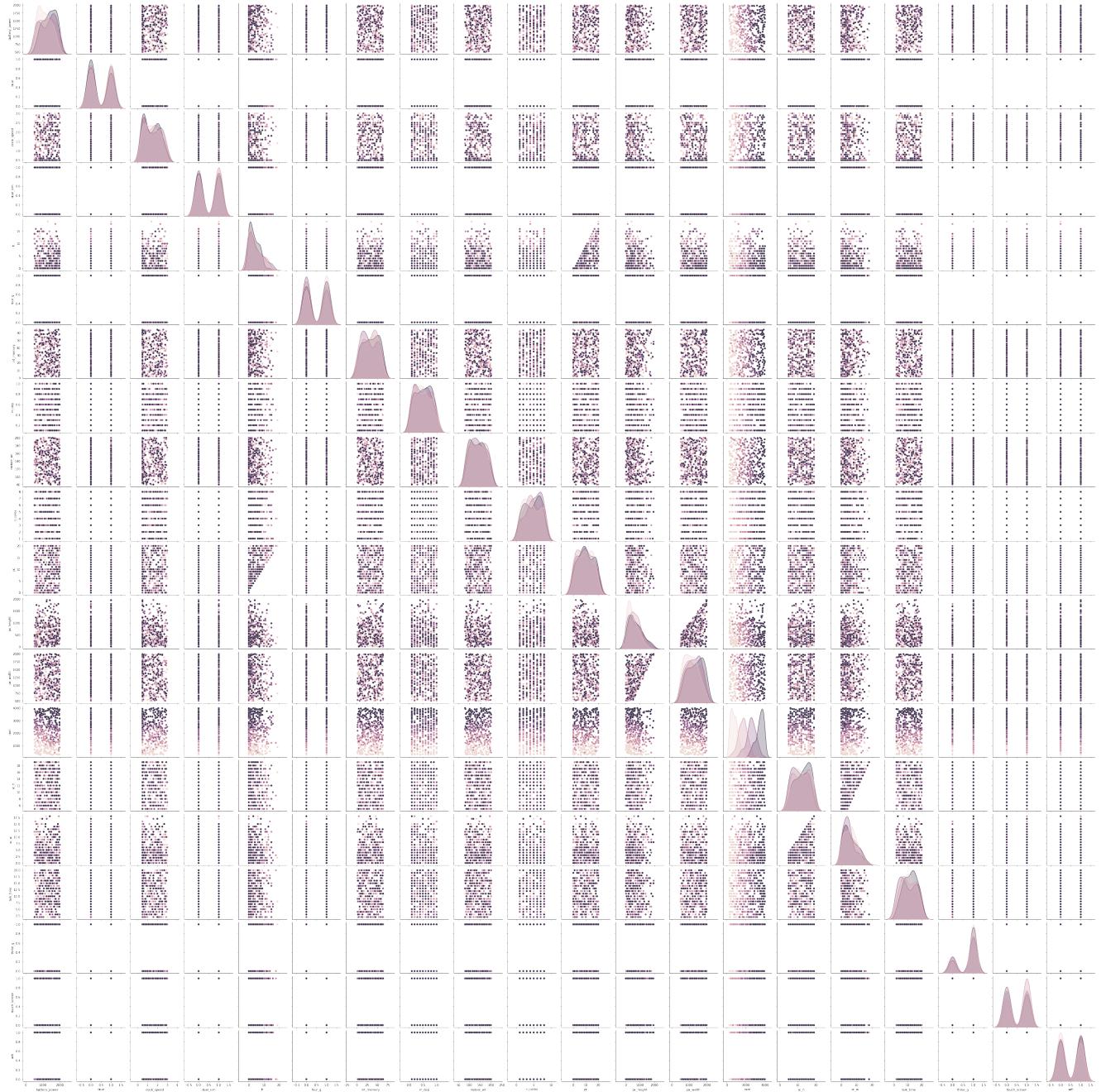
1. liściem według kryterium stopu – kończymy to wywołanie rekurencyjne;
2. węzłem rozgałęziającym się według kryterium wyboru atrybutu – dokonujemy wyboru atrybutu, tworzymy rozgałęzienia według wartości, jakie przyjmuje dany atrybut, i dla każdego węzła potomnego tworzymy rekurencyjne wywołanie algorytmu, z listą atrybutów zmniejszoną o właśnie wybrany atrybut.

Wszystkie algorytmy działają według podanego schematu, różnice w implementacji dotyczą kryteriów stopu i wyboru atrybutu.

2.2 Analiza danych

Pairplot to wykres par, który pozwala nam zobaczyć zarówno rozkład pojedynczych zmiennych, jak i relacje między dwiema zmiennymi.

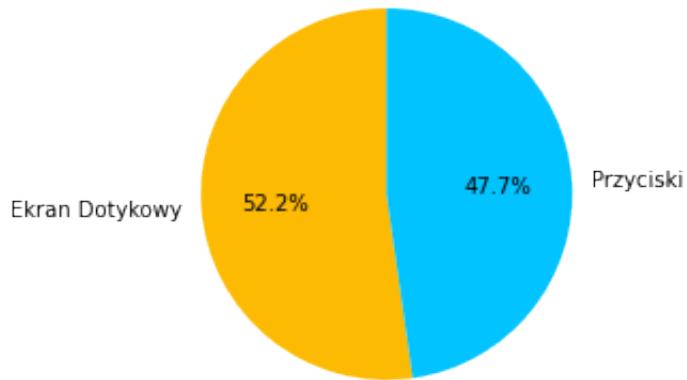
Wizualizacja zbioru danych



Na naszym wykresie par, dla zmiennych, gdzie cena jest najniższa - kolor jest najjaśniejszy, a im cena jest wyższa, tym kolor jest ciemniejszy. Najbardziej klarowny i czytelny rozkład otrzymaliśmy dla zmiennych odpowiadających ilości RAMu w urządzeniach z bazy. W wykresach, gdzie RAM jest osią X-ów lub osią Y-ów rzucają się w oczy gradienty charakteryzujące zależność ceny telefonu od ilości pamięci RAM.

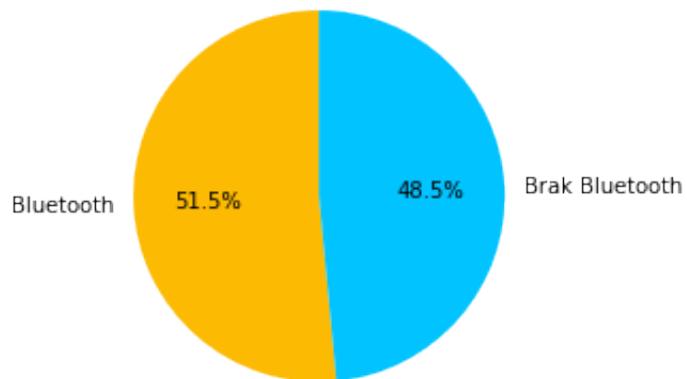
Wykres kołowy to wykres statystyczny, na którym można wyświetlić tylko jedną serię danych. Obszar wykresu to całkowity procent podanych danych. Obszar fragmentów koła reprezentuje procent części danych.

Procent urządzeń z ekranem dotykowym



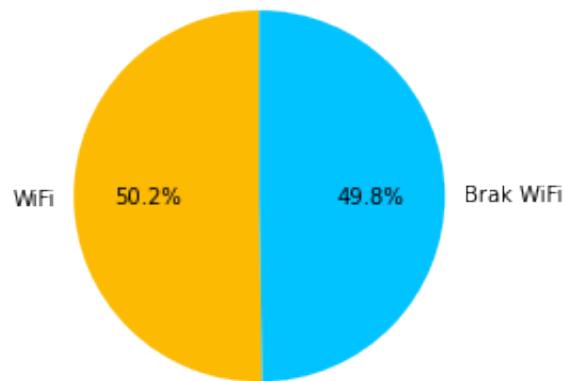
Powyższy wykres ukazuje, iż w użytej przez nas próbce badawczej 52,2% urządzeń posiada wbudowany ekran dotykowy, a pozostałe 47,7% posiada przyciski.

Procent urządzeń z Bluetooth



Na powyższym wykresie widnieje rozkład procentowy ilości telefonów z dostępem do sieci Bluetooth w badanej przez nas części bazy danych. Większość telefonów - 51,5% - posiada wymieniony standard, a 48,5% nie ma do niego dostępu.

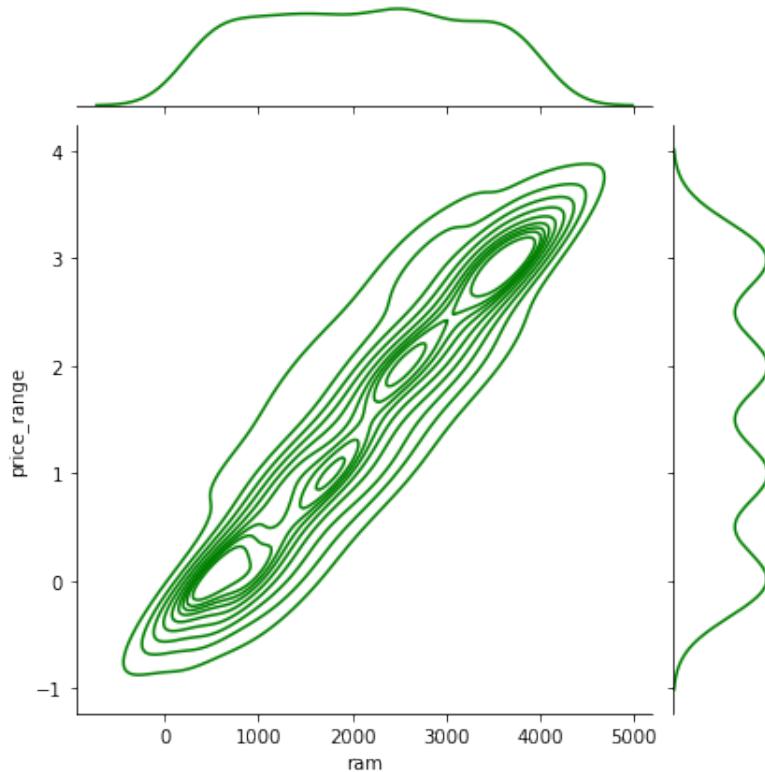
Procent urządzeń z WiFi



Powyższy wykres przedstawia rozkład procentowy ilości telefonów z dostępem do sieci WiFi w badanej części zbioru danych. Większość (50,2%) posiada możliwość połączenia się z WiFi, ale różnica między oboma podzbiorami jest mała i wynosi zaledwie 0,4%.

Jointplot pokazuje zależność między 2 zmiennymi (dwuwymiarowymi) oraz profilami jednowymiarowymi na marginesach. W tym przypadku wykres jest typu - „kde”, czyli określa oszacowanie gęstości jądra na marginesach i przekształca wnętrze na wykres konturowy.

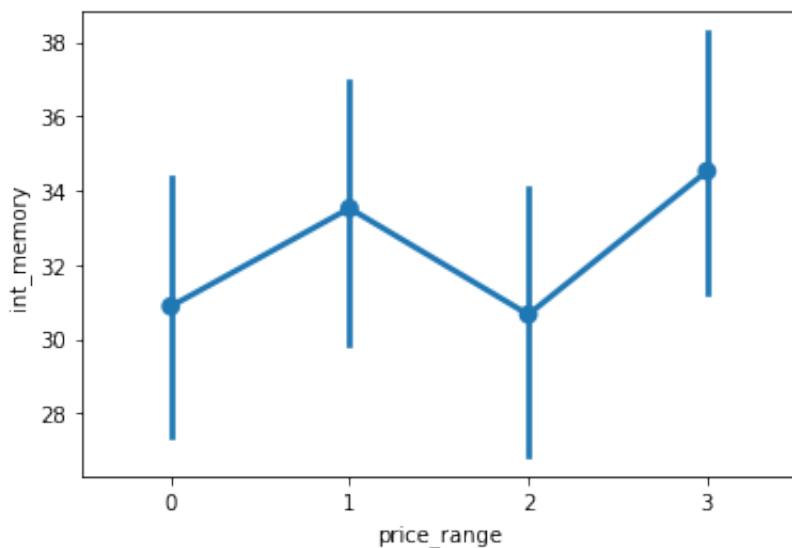
Zależność ceny od RAMu



Powyższy wykres idealnie obrazuje zależność pomiędzy RAMem a ceną. Cena telefonu zwiększa się wraz z przyrostem ilości pamięci RAM - im droższy telefon tym więcej tego zasobu posiada.

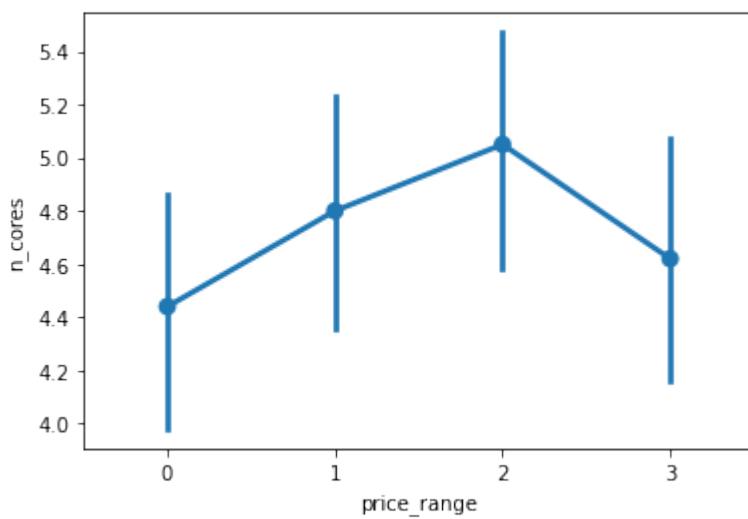
Wykres pointplot jest graficzną wizualizacją danych, która przedstawia oszacowania punktowe dla kategorii o określonych przedziałach ufności.

Zależność ceny od ilości pamięci



Na umieszczonym wyżej wykresie za pomocą kropek określona jest średnia ilość pamięci - dla każdej z cen, a linie pionowe oznaczają przedziały ufności. Wartości dla ceny niskiej (0) i wysokiej (2) są bardzo porównywalne. Nie możemy przez to jasno stwierdzić, czy ilość pamięci telefonu ma mniejsze znaczenie przy ustalaniu jego ceny - co mogłoby być trochę nie logicznym sformuowaniem, gdyż urządzenie z większą ilością pamięci powinno być droższe; czy po prostu jest to efekt losowo wybranych rekordów w badanej próbce.

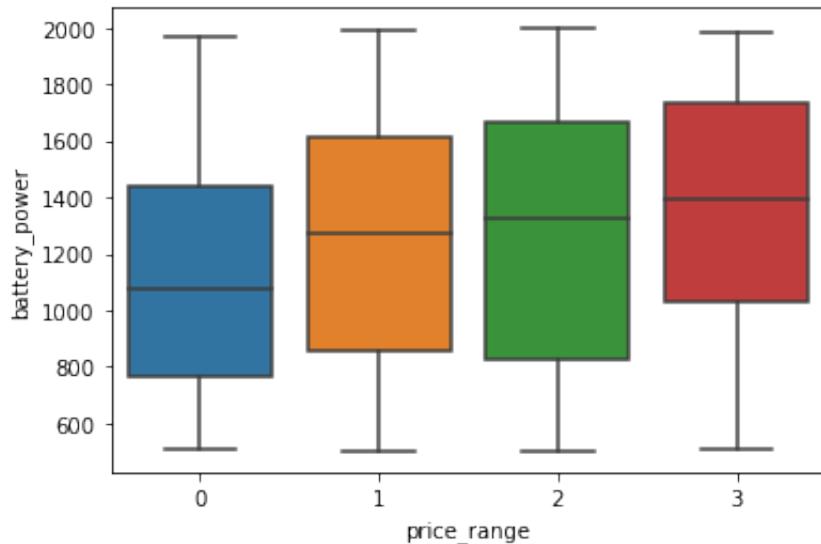
Zależność ceny od ilości rdzeni



Na podstawie powyższego wykresu można zauważyć, że średnia ilość rdzeni wykazuje tendencję rosnącą wraz ze zwiększaniem się ceny telefonu. Mniejsza średnia ilość rdzeni w przypadku telefonów o cenie bardzo wysokiej (w stosunku do telefonów o cenie średniej i wysokiej) jest prawdopodobnie spowodowana tym, iż grupa urządzeń opatrzona nr 3 ma rdzenie bardziej zaawansowane przez co może ich znajdować się zwyczajnie mniej.

Wykres pudełkowy to sposób na pokazanie na wykresie podsumowania pięciu liczb. Główna część wykresu („ramka”) pokazuje, gdzie znajduje się środkowa część danych: przedział międzykwartylowy. Na końcach ramki znajduje się pierwszy kwartyl (oznaczenie 25%) i trzeci kwartyl (oznaczenie 75%). Dolny "wąs" to minimum (najmniejsza liczba w zestawie), a górny to maksimum (największa liczba w zestawie). Mediana jest reprezentowana przez poziomą kreskę pośrodku pudełka.

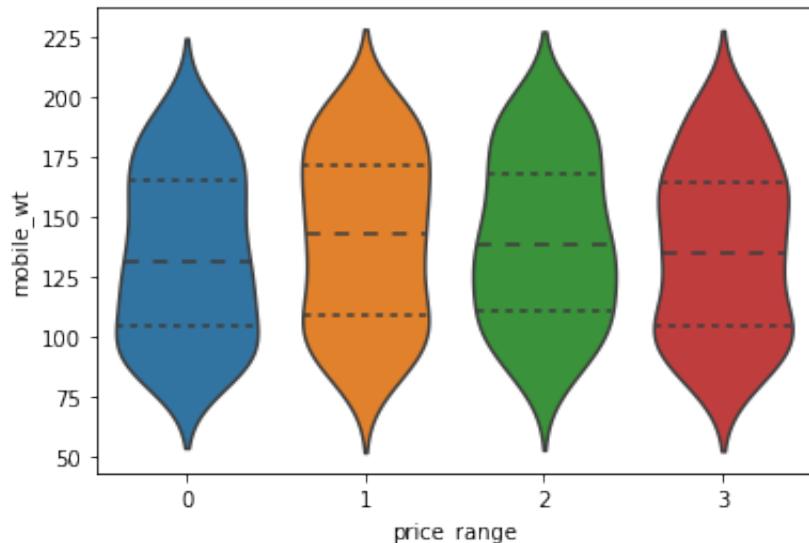
Zależność ceny od pojemności baterii



Na zaprezentowanym powyżej wykresie widać, że pojemność baterii i długość jej działania ma wpływ na cenę telefonu. Odzwierciedla to mediana dla pojemności baterii, która znajduje się tym wyżej, im cena telefonu jest wyższa.

Wykres wiolinowy jest niezwykle podobny do przedstawienia pudełkowego - to również sposób na pokazanie na wykresie podsumowania pięciu liczb. Na samej górze znajduje się wartość maksymalna, następnie trzeci kwartyl (oznaczenie 75%), później mediana i kwartyl pierwszy (oznaczenie 25%), na samym dole znajduje się minimum.

Zależność ceny od wagi



Na podstawie powyższego wykresu możemy stwierdzić, że waga nie ma większego znaczenia dla ceny urządzenia, gdyż mediany wszystkich 4 elementów wykresu znajdują się na w miarę porównywalnym poziomie.

2.3 Algorytmy

Algorytm KNN miał za zadanie wyznaczyć ceny dla rekordów ze zbioru testowego (zbior testowy to 0.3 wszystkich rekordów z bazy), gdzie do wyliczania odległości została użyta metryka Minkowskiego - dla której stopień pierwiastka 'p' ustaliliśmy na 2. Na podstawie uprzednio uzyskanych klasyfikacji, została wyznaczona dokładność z jaką klasyfikatorowi udało się wyznaczyć poprawne ceny, w porównaniu do faktycznych.

Algorithm 1 K-nearest neighbours algorithm.

Data: $X_train, y_train, X_test, y_test, k$
foreach point in test data **do**
 find the distance to all training data points;
 compute the distances;
 sort the computed distances;
 select k-nearest reference points corresponding to k smallest distances;
end

Result: $knn.score$

Algorytm Drzew Decyzyjnych miał za zadanie wyznaczyć ceny dla rekordów ze zbioru testowego (zbior testowy to 0.3 wszystkich rekordów z bazy), gdzie do klasyfikacji została użyta metoda podejmowania decyzji prawda-fałsz na podstawie warunków. Przewidywania to liście (już dalej nie podzielne elementy) - zbiory lub pojedyńczy element danej ceny, których nie da się dalej rozdzielić na ceny odmienne. Na podstawie uprzednio uzyskanych klasyfikacji, została wyznaczona dokładność z jaką klasyfikatorowi udało się wyznaczyć poprawne ceny, w porównaniu do faktycznych.

Algorithm 2 Decission tree algorithm.

Data: $X_train, y_train, X_test, y_test$
 $dtree(tree, test_point);$
if tree is of the form Leaf(guess) **then**
 return guess;
end
if tree is of the form Node(f, right, left) **then**
 if $f = no$ in the test_point **then**
 return dtree(left, test_point)
 else
 return dtree(right, test_point)
 end
end

Result: $dtree.score$

2.4 Implementacja

Do klasyfikacji użyliśmy KNN i Drzew Decyzyjnych ze zewnętrznej biblioteki *scikit learn*. Do rozwiązania zadania potrzebne były takie funkcje/konstruktory, jak:

- KNeighborsClassifier(),
- fit(),
- predict(),
- score().

dla algorytmu KNN.

```
1 # w naszej klasyfikacji uzyliśmy metryki minkowskiego - znanej nam już
2 # ze zajec, o stopniu pierwiastka 'p' równym 2
3 # liczba sąsiadów jest najlepsza, która wychodzi nam z analizy
4 # error_rate
5 knn = KNeighborsClassifier(n_neighbors = k, metric='minkowski', p = 2)
6
7 # metoda fit() przyjmuje dane uczące jako argumenty, które sa
8 # dwiema tablicami w przypadku uczenia nadzorowanego
9 # funkcja fit() dostosowuje wagi zgodnie z wartościami danych,
10 # dzięki czemu można uzyskać lepszą dokładność.
11 # Wielo w skrócie dopasowanie to trening. Następnie,
12 # po przeszkoleniu, model może być używany do tworzenia prognoz,
13 # zwykle z wywołaniem metody .predict() lub .score()
14 knn.fit(X_train,y_train)
15
16 # metoda score()wróci dokładność podanych danych testowych
17 # oraz etykiet - faktycznych wyników
18 knn.score(X_test,y_test)
19
20 # przewiduje wyniki - ceny na podstawie klasyfikatora
21 knn.predict(X_test)
```

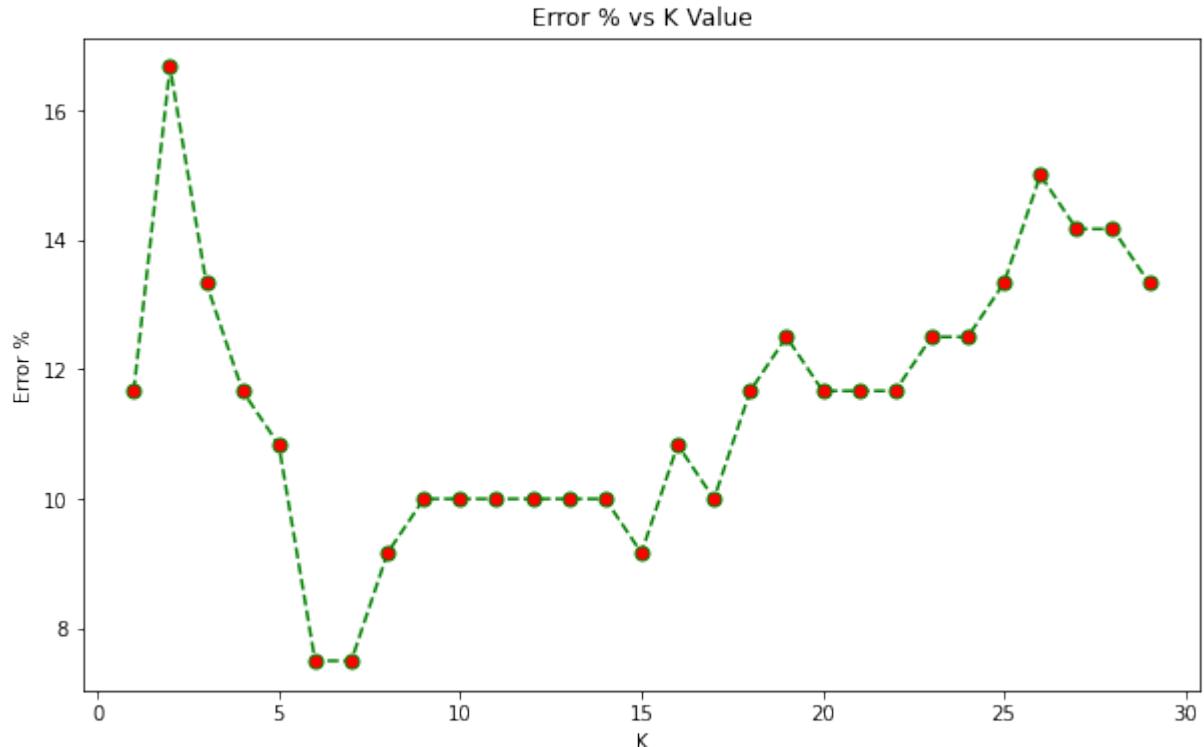
Do wyznaczenia najbardziej optymalnego K użyliśmy poniższego kodu:

```
1 from sklearn.neighbors import KNeighborsClassifier
2
3 # tablica na przechowywanie % poziomu error_rate
4 error_rate = []
5
6 # dla każdego 'k' w przedziale od 1 do 30
7 for i in range(1,30):
8     # tworzymy klasyfikator
9     knn = KNeighborsClassifier(n_neighbors = i)
10    knn.fit(X_train,y_train)
11    # przewidujemy ceny dla zbioru testowego
12    pred_i = knn.predict(X_test)
13    # następnie przewidziane ceny porównujemy do faktycznych
14    # i do tablicy wrzucamy % error_rate
15    error_rate.append(np.mean(pred_i != y_test) * 100)
16
17 # na koncu wybieramy 'k', dla którego blad jest najmniejszy
```

```
18 k = np.argmin(error_rate) + 1
```

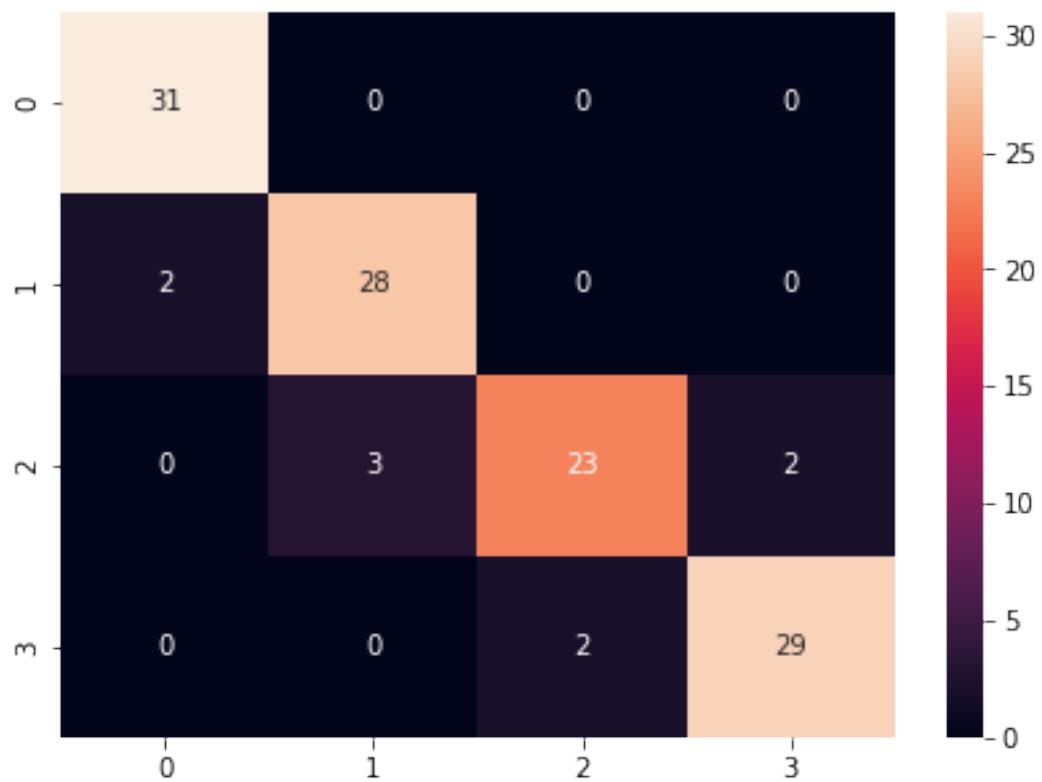
W drzewach decyzyjnym metody pełnią analogiczną funkcję.

2.5 Analiza wyników

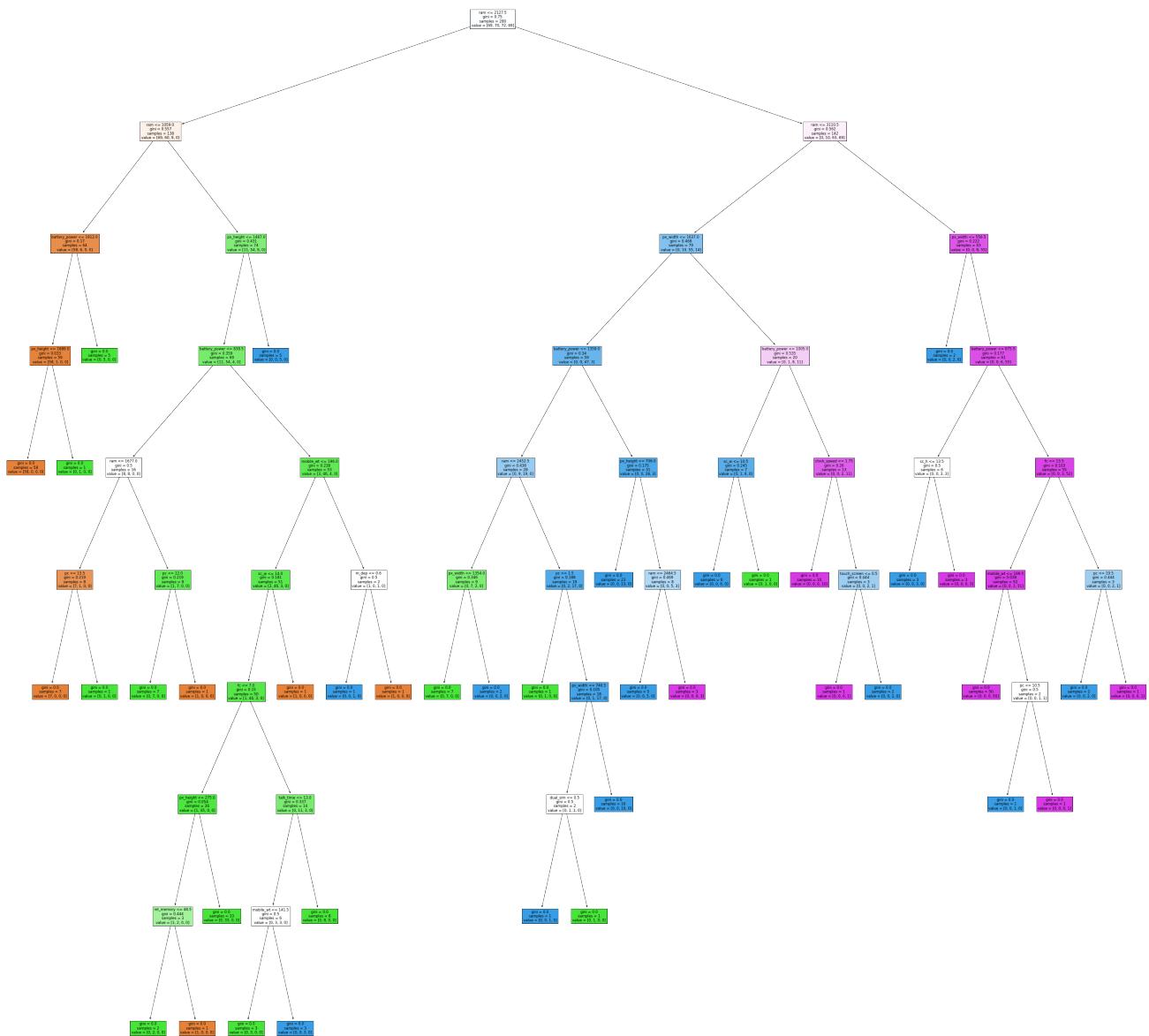


Dla K należącego do $<1:30)$ tworzymy klasyfikatory KNN, a następnie porównujemy przewidywane przez nie wyniku do faktycznych wartości. Z porównanych wyników tworzymy procentowy $error_rate$ i dla każdej wartości K nanosimy na wykres odpowiadający jej $error_rate$. Najlepsza ilość sąsiadów K to taka, dla której $error_rate$ jest najmniejszy. Na powyższym wykresie widać, że najlepsza dokładność wyjdzie dla $k = 6$ oraz $k = 7$. Po sklasyfikowaniu naszego zbioru KNN, używając $k = 6$ dokładność wyniosła: 92.5%.

Poniżej znajduje się heatmapa (mapa cieplna) przedstawiająca skuteczność klasyfikacji KNN dla zbioru testowego. Na przekątnej znajdują się dobrze zaklasyfikowane rekordy - to jest 92.5% skuteczności, a poza przekątną znajdują się te, których cena została źle przewidziana.



Decission tree



Adekwatnie do nazwy, przewidywanie cen jest prowadzone drogą decyzji, aż do momentu uzyskania liści (gdy część elementów danej ceny zostanie odizolowana od elementów innych cen, wtedy dalsze dzielenie zbioru nie jest potrzebne), gdzie poszczególne kolory oznaczają:

- **pomarańczowy** - cenę niską,
- **zielony** - cenę średnią,
- **niebieski** - cenę wysoką,
- **fioletowy** - cenę najwyższą.

Otrzymana dokładność dla drzewa decyzyjnego jest równa: 80%, zatem jest mniejsza niż w przypadku KNN'a.

2.6 Podsumowanie

Wyniki klasyfikacji:

- KNN: 92,5%
- Drzewo decyzyjne: 80%

Na podstawie otrzymanych wyników możemy stwierdzić, iż algorytm K-najbliższych sąsiadów był lepszym klasyfikatorem dla przyjętej przez nas próbki bazy danych, gdyż miał on większą skuteczność od algorytmu drzew decyzyjnych.

Przy zmianie wielkości próbki (liczba rekordów inna niż 400) wyniki różnią się o kilka procent, ale w większości przypadków lepsze rezultaty dalej osiąga KNN. W niektórych przypadkach wyniku obu klasyfikatorów były bardziej zbliżone, lecz nie na tyle, aby stwierdzić, że decision tree mogłoby wziąć góre nad KNN'em.

3 Pełen kod aplikacji

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import random as rd
6
7 data_set = pd.read_csv('train.csv')
8 data_set = data_set.sort_values(by = 'price_range')
9
10 # redukcja z 2000 rekordow do 400, zachowujac proporcje wystapien
11 # telefonow danej ceny
12 dataset = data_set[400:500]
13 dataset = dataset.append(data_set[900:1000])
14 dataset = dataset.append(data_set[1400:1500])
15 dataset = dataset.append(data_set[1900:2000])
16
17 dataset.head()
18 dataset.info()
19 dataset.describe()
20
21 sns.pairplot(dataset,
22               hue='price_range',
23               kind = 'scatter'
24               )
25
26 sns.jointplot(x = 'ram',
27                 y = 'price_range',
28                 data = dataset,
29                 color = 'green',
30                 kind='kde')
31
32 sns.pointplot(
33                 y="int_memory",
34                 x="price_range",
35                 data=dataset
36                 )
37
38 fig1, ax1 = plt.subplots()
39 ax1.pie(
40         dataset['touch_screen'].value_counts().values,
41         labels = ["Ekran Dotykowy","Przyciski"],
42         colors = ['#fcba03', '#00c3ff'],
43         autopct = '%1.1f%%',
44         startangle = 90)
45
46 plt.show()
47
48 fig1, ax1 = plt.subplots()
49 ax1.pie(
50         dataset['blue'].value_counts().values,
51         labels = ["Bluetooth","Brak Bluetooth"],
52         colors = ['#fcba03', '#00c3ff'],
53         autopct = '%1.1f%%',
```

```

52         startangle = 90)
53 plt.show()
54
55 fig1, ax1 = plt.subplots()
56 ax1.pie(
57     dataset['wifi'].value_counts().values,
58     labels = ["WiFi", "Brak WiFi"],
59     colors = ['#fcba03', '#00c3ff'],
60     autopct = '%1.1f%%',
61     startangle = 90)
62 plt.show()
63
64 sns.boxplot(
65     x="price_range",
66     y="battery_power",
67     data=dataset
68 )
69
70 sns.violinplot(x='price_range',
71                  y='mobile_wt',
72                  data=dataset,
73                  inner='quartile')
74
75 sns.pointplot(
76     y="n_cores",
77     x="price_range",
78     data=dataset
79 )
80
81 # X to zbior bez 'price_range'
82 X = dataset.drop('price_range',
83                   axis = 1
84 )
85 X
86
87 # zbior y zawiera same 'price_range'
88 y = dataset['price_range']
89 y
90
91 from sklearn.model_selection import train_test_split
92 # przecina X i y na zbiory treningowe i testowe,
93 # zbiory przed cieciem sa tasowane
94 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
95 0.3)
96
97 from sklearn.neighbors import KNeighborsClassifier
98 error_rate = []
99 for i in range(1,30):
100     knn = KNeighborsClassifier(n_neighbors = i)
101     knn.fit(X_train,y_train)
102     pred_i = knn.predict(X_test)
103     error_rate.append(np.mean(pred_i != y_test) * 100)
104 plt.figure(figsize=(10,6))
105 plt.plot(

```

```

106         range(1,30),
107         error_rate,
108         color='green',
109         linestyle='dashed',
110         marker='o',
111         markerfacecolor='red',
112         markersize = 7
113     )
114 plt.title('Error % vs K Value')
115 plt.xlabel('K')
116 plt.ylabel('Error %')
117 print("Najlepsze K: ", np.argmin(error_rate) + 1)
118 k = np.argmin(error_rate) + 1
119
120 knn = KNeighborsClassifier(n_neighbors = k, metric='minkowski', p = 2)
121 knn.fit(X_train,y_train)
122 knn.score(X_test,y_test)
123
124 X_train
125 X_test
126 y_train
127 y_test
128
129 from sklearn.tree import DecisionTreeClassifier
130 from sklearn import tree
131 dtree = DecisionTreeClassifier()
132 dtree.fit(X_train,y_train)
133 dtree.score(X_test,y_test)
134 fig = plt.figure(figsize=(50,50))
135 _ = tree.plot_tree(dtree,
136                     feature_names = ['battery_power', 'blue', ,
137                                     'clock_speed', 'dual_sim', 'fc', 'four_g',
138                                     'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
139                                     'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
140                                     'touch_screen', 'wifi'],
141                                     filled=True
142                     )
143
144 from sklearn.metrics import classification_report,confusion_matrix
145 pred = knn.predict(X_test)
146 matrix = confusion_matrix(y_test, pred)
147 print(matrix)
148 plt.figure(figsize = (7,5))
149 sns.heatmap(matrix,
150             annot = True)

```

Spis treści

1	Część I	1
1.1	Opis projektu	1
1.2	Baza danych	1
1.3	Dodatkowe informacje	1
2	Część II	2
2.1	Opis działania	2
2.2	Analiza danych	3
2.3	Algorytmy	10
2.4	Implementacja	11
2.5	Analiza wyników	12
2.6	Podsumowanie	15
3	Pełen kod aplikacji	16