# Analiza eksploracyjna oraz klasyfikacja przy użyciu metod uczenia zespołowego na zbiorze danych Titanic

Damian Lewańczyk

## Spis treści



## 1 Analiza eksploracyjna

### 1.1 Opis danych

Nasz zbiór danych `Titanic` złożony jest z 1309 obserwacji - pasażerów słynnego statku. Zawiera 14 różnych zmiennych objaśniających opisujących poszczególne osoby oraz zmienną objaśnianą `survived` - opisuje ona czy dany człowiek przeżył (`survived` = 1) czy nie (`survived` = 0). Przed analizą danych przedstawimy krótki opis każdej zmiennej, przedstawiając jej nazwę oraz typ:

- **pclass** (zmienna jakościowa, przyjmuje 3 różne wartości) - określa klasę biletu pasażera
- **survived** (zmienna jakościowa, przyjmuje 2 różne wartości) - zmienna objaśniana, opisuje czy dany pasażer przeżył katastrofę czy nie

- **name** (zmienna jakościowa, przyjmuje 1307 różnych wartości) - imię i nazwisko pasażera
- **sex** (zmienna jakościowa, przyjmuje 2 różne wartości) - płeć pasażera
- **age** (zmienna ilościowa, przyjmuje wartości w zakresie od 0.1667 do 80) - wiek osoby
- **sibsp** (zmienna jakościowa, przyjmuje 7 różnych wartości) - liczba rodzeństwa i współmałżonków danej osoby, którzy też znajdują się na pokładzie statku
- **parch** (zmienna jakościowa, przyjmuje 8 różnych wartości) - liczba dzieci i rodziców danej osoby znajdujących się na statku
- **ticket** (zmienna jakościowa, przyjmuje 929 różnych wartości) - numer biletu
- **fare** (zmienna ilościowa typu ciągłego, przyjmuje wartości w zakresie od 0 do 512.3292) - opłata za podróż
- **cabin** (zmienna jakościowa, przyjmuje 186 różnych wartości) - kabina danego pasażera
- **embarked** (zmienna jakościowa, przyjmuje 3 różne wartości) - port z którego wyruszył dany pasażer: C = Cherbourg; Q = Queenstown; S = Southampton
- **boat** (zmienna jakościowa, przyjmuje 27 różnych wartości) - numer łodzi, jeśli pasażer przeżył
- **body** (zmienna jakościowa, przyjmuje 121 różnych wartości) - numer ciała, jeśli pasażer nie przeżył, a ciało odnaleziono
- **home.dest** (zmienna jakościowa, przyjmuje 369 różnych wartości) - miasto do którego udawał się pasażer

Dodatkowo, w tabelach 1 i 2 przedstawione jest pierwsze 10 wierszy zbioru danych, podzielonych na dwie części po 7 cech.

Tabela 1: Pierwsze 10 wierszy zbioru danych Titanic, kolumny 1-7

| pclass | survived | name | sex | age | sibsp | parch |
|---|---|---|---|---|---|---|
| 1 | 1 | ALLEN, MISS. ELISABETH WALTON | female | 29 | 0 | 0 |
| 1 | 1 | ALLISON, MASTER. HUDSON TREVOR | male | 0,9167 | 1 | 2 |
| 1 | 0 | ALLISON, MISS. HELEN LORAINE | female | 2 | 1 | 2 |
| 1 | 0 | ALLISON, MR. HUDSON JOSHUA CREIGHTON | male | 30 | 1 | 2 |
| 1 | 0 | ALLISON, MRS. HUDSON J C (BESSIE WALDO DANIELS) | female | 25 | 1 | 2 |
| 1 | 1 | ANDERSON, MR. HARRY | male | 48 | 0 | 0 |
| 1 | 1 | ANDREWS, MISS. KORNELIA THEODOSIA | female | 63 | 1 | 0 |
| 1 | 0 | ANDREWS, MR. THOMAS JR | male | 39 | 0 | 0 |
| 1 | 1 | APPLETON, MRS. EDWARD DALE (CHARLOTTE LAMSON) | female | 53 | 2 | 0 |
| 1 | 0 | ARTAGAVEYTIA, MR. RAMON | male | 71 | 0 | 0 |

Tabela 2: Pierwsze 10 wierszy zbioru danych Titanic, kolumny 8-14

| ticket | fare | cabin | embarked | boat | body | home.dest |
|---|---|---|---|---|---|---|
| 24160 | 211,3375 | B5 | S | 2 | NaN | St Louis, MO |
| 113781 | 151,5500 | C22 C26 | S | 11 | NaN | Montreal, PQ / Chesterville, ON |
| 113781 | 151,5500 | C22 C26 | S | NaN | NaN | Montreal, PQ / Chesterville, ON |
| 113781 | 151,5500 | C22 C26 | S | NaN | 135.0 | Montreal, PQ / Chesterville, ON |
| 113781 | 151,5500 | C22 C26 | S | NaN | NaN | Montreal, PQ / Chesterville, ON |
| 19952 | 26,5500 | E12 | S | 3 | NaN | New York, NY |
| 13502 | 77,9583 | D7 | S | 10 | NaN | Hudson, NY |
| 112050 | 0,0000 | A36 | S | NaN | NaN | Belfast, NI |
| 11769 | 51,4792 | C101 | S | D | NaN | Bayside, Queens, NY |
| PC 17609 | 49,5042 | NaN | C | NaN | 22.0 | Montevideo, Uruguay |

## 1.2 Szukanie i uzupełnianie brakujących wartości

Użyjemy metody `isnull()` dla obiektu klasy **pandas.DataFrame**, aby sprawdzić ile wartości brakujących ma każda zmienna oraz jaki jest to odsetek całości. W kodzie źródłowym 1 podane zostały bezwzględne liczby wartości brakujących dla poszczególnych zmiennych, a w kodzie źródłowym 2 procentowy udział tych wartości w stosunku do wszystkich wierszy.

```
Out[103]:
pclass          0
survived        0
name            0
sex             0
age           263
sibsp           0
parch           0
ticket          0
fare            1
cabin        1014
embarked        2
boat          823
body         1188
home.dest     564
```

Kod źródłowy 1: Liczba wartości brakujących dla każdej cechy

```
Out[113]:
pclass        0.0
survived      0.0
name          0.0
sex           0.0
age          20.1
sibsp         0.0
parch         0.0
ticket        0.0
fare          0.1
cabin        77.5
embarked      0.2
```

```
boat          62.9
body          90.8
home.dest     43.1
```

Kod źródłowy 2: Odsetek wartości brakujących dla każdej cechy

### 1.2.1 Wytypowanie i usuwanie niepotrzebnych zmiennych

Widzimy powyżej, że zmienne `cabin`, `boat`, `body` i `home.dest` mają najwięcej brakujących wartości, każda z nich ponad 40%, tym samym usuwamy je z modelu. Ponadto, usuwamy zmienne z imieniem i nazwiskiem oraz numerem biletu pasażera, ponieważ oczywiście nie mają one wpływu na potencjalne przeżycie danej osoby (te zmienne jakościowe, czyli `name` i `ticket` mają bardzo dużo kategorii). W celu usunięcia zmiennych posłużymy się metodą `drop()` dla obiektu klasy **pandas.DataFrame**, co widzimy w kodzie źródłowym 3. Jednakże, przed usunięciem zmiennej `name`, dodamy na jej bazie zmodyfikowaną wersję - zmienną `initial`, która odpowiada tytułowi danej osoby ("Mr.", "Mrs." itd.), jednakże grupujemy tą zmienną na 5 kategorii: "Master", "Miss", "Mr", "Mrs" i "Other". Zmienną `initial` wykorzystamy tylko i wyłącznie w celu trafniejszemu przypisaniu wartości brakujących zmiennej `age`, a później ją usuniemy.

```python
df_titanic['initial']=0
df_titanic["initial"] = df_titanic["name"].apply(lambda st: st[st.find(",")+2:st.find(".")])
df_titanic['initial'].replace(['Mlle','Mme','Ms','Dr','Major','Lady','Countess', 'Jonkheer','Col','Rev','Capt','Sir','Don'],['Miss','Miss','Miss','Mr','Mr','Mrs','Mrs','Other','Other','Other','Mr','Mr','Mr'], inplace=True)
df_titanic = df_titanic.drop(["cabin", "boat", "body", "home.dest", "name", "ticket"], axis=1)
```

Kod źródłowy 3: Usuwanie niepotrzebnych zmiennych z modelu

### 1.2.2 Przypisanie typów zmiennych

Przed uzupełnieniem wartości brakujących, musimy przypisać zmiennym `age` i `fare` typ numeryczny. Zrobimy to za pomocą funkcji `pandas.to_numeric()`.

```python
df_titanic['age'] = df_titanic['age'].dropna().str.replace(",", ".")
df_titanic['fare'] = df_titanic['fare'].dropna().str.replace(",", ".")
df_titanic[['age', 'fare']] = df_titanic[['age','fare']].apply(pd.to_numeric)
```

Kod źródłowy 4: Przypisanie typu zmiennym ilościowym

### 1.2.3 Wypełnianie brakujących wartości

Jeśli chodzi o brakujące wartości zmiennych `age`, `fare` oraz `embarked`, wypełnimy brakujące wartości w następujący sposób:
- przy zmiennej jakościowej `embarked`, wartości brakujące zastąpimy najczęściej występującą wartością, czyli "S"
- jedną brakującą wartość zmiennej ilościowej `fare` zastąpimy średnią z postałych wartości
- w zmiennej `age` wartości brakujące zastąpimy średnimi wartościami dla danej grupy zmiennej `initial`

Na samym końcu usuwamy niepotrzebną już nam zmienną `initial`.

```
df_titanic["embarked"] = df_titanic["embarked"].fillna(df_titanic['embarked'].mode())
df_titanic["fare"] = df_titanic["fare"].fillna(df_titanic['fare'].mean())
df_titanic['age'] = df_titanic['age'].fillna(df_titanic.groupby('initial')['age'].transform('mean'))
df_titanic = df_titanic.drop("embarked", axis=1)
```
Kod źródłowy 5: Przypisanie typu zmiennym ilościowym

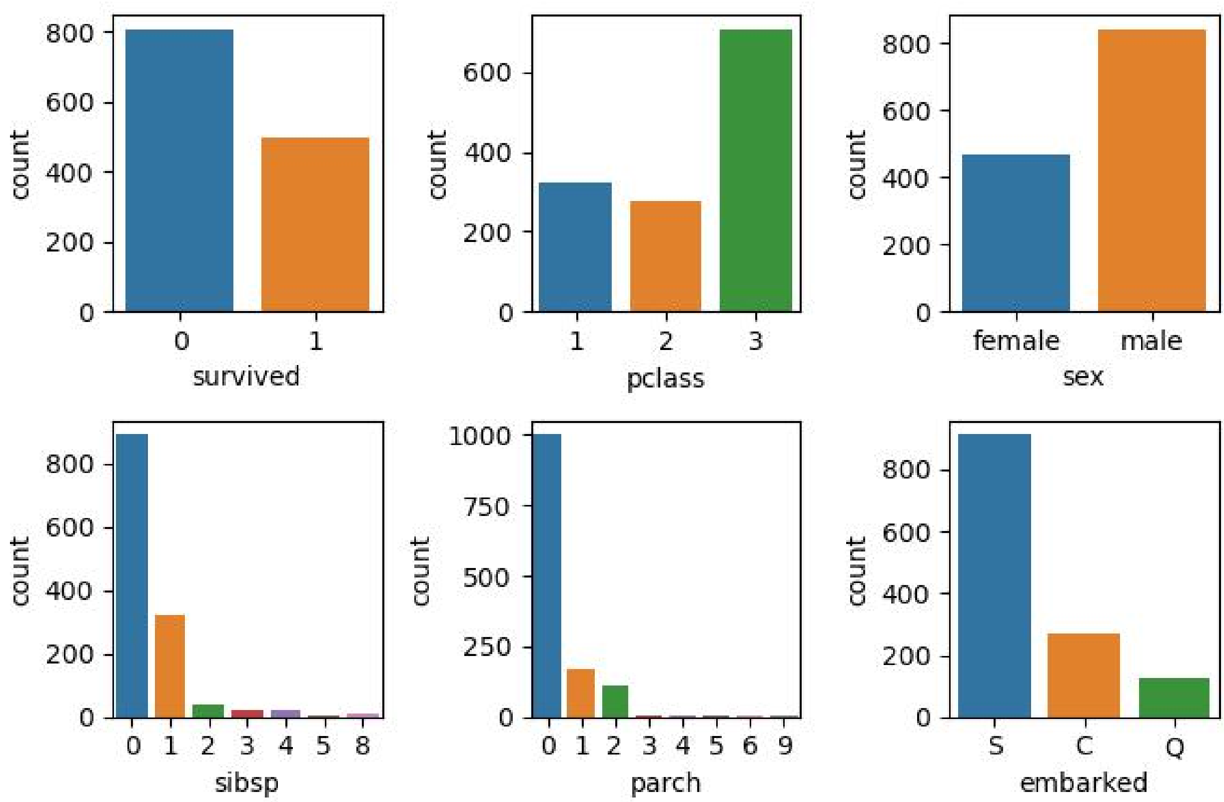
Ostatecznie w naszym modelu zostało 7 zmiennych objaśniających: 5 jakościowych i 2 ilościowe oraz jakościowa zmienna objaśniana `survived`.

Tabela 3: Pierwsze 5 wierszy zbioru danych Titanic po modyfikacjach

| survived | pclass | sex | age | sibsp | parch | fare | embarked |
|---|---|---|---|---|---|---|---|
| 1 | 1 | female | 29 | 0 | 0 | 211,3375 | S |
| 1 | 1 | male | 0,9167 | 1 | 2 | 151,5500 | S |
| 0 | 1 | female | 2 | 1 | 2 | 151,5500 | S |
| 0 | 1 | male | 30 | 1 | 2 | 151,5500 | S |
| 0 | 1 | female | 25 | 1 | 2 | 151,5500 | S |

## 1.3 Wizualizacja danych

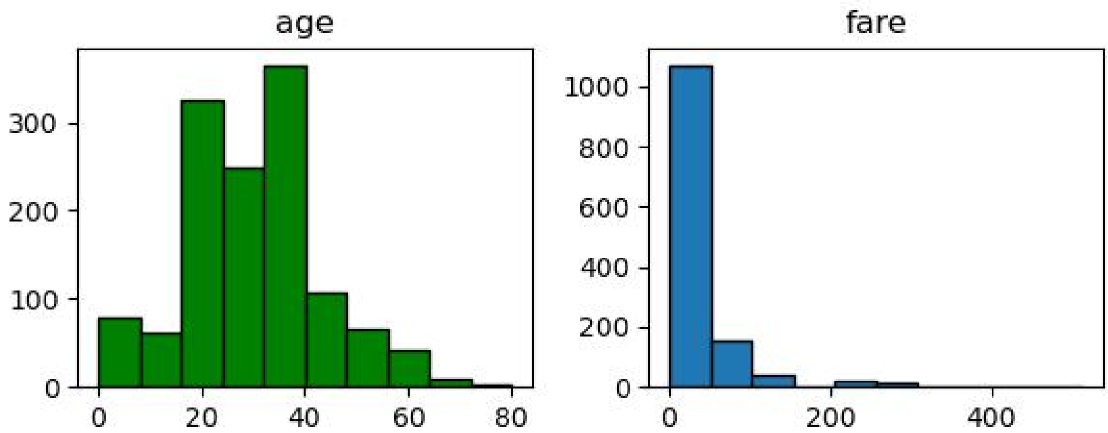
W tym podrozdziale przeprowadzona zostanie wizualizacja danych. Na początku, zaprezentujemy wykresy słupkowe dla zmiennych jakościowych, które przedstawią częstości występowania poszczególnych kategorii dla każdej zmiennej typu factor. Zrobimy to za pomocą funkcji `countplot` [1] z pakietu **seaborn**, a wyniki przedstawione są na rysunku 1.




Rysunek 1: Częstotliwości występowania zmiennych jakościowych

Jak widzimy powyżej, rozkłady kategorii poszczególnych cech jakościowych oczywiście różnią się od siebie. Możemy rozróżnić trzy dwuelementowe grupy: zmienne `survived` i `sex`, mające po 2 kategorie, mają najbardziej równomierne rozkłady, ale wciąż z wyraźną przewagą występowania jednej kategorii: wartość **0** dla zmiennej `survived` oraz **male** dla zmiennej `sex` - kategorie te to około 60% wszystkich wartości odpowiednich zmiennych. Następnie, zmienne `pclass` i `embarked`, mające po 3 kategorie, mają jedną wartość występującą dużo częściej od pozostałych: `pclass`=**3** i `embarked`=**S**. Natomiast zmienne `sibsp` i `parch` mają więcej kategorii - odpowiednio 7 i 8. Również i w wypadku tych zmiennych widzimy jedną wartość wyraźnie dominującą jeśli chodzi o częstotliwość występowania, jest to 0 dla obu tych zmiennych.

Poniżej przedstawione zostały histogramy dla zmiennych numerycznych. Stworzone zostały za pomocą metody `.hist()` [2] dla obiektu klasy **matplotlib.Axis**. Rezultaty przedstawione zostały na rysunku 2.




Rysunek 2: Histogramy dla zmiennych ilościowych

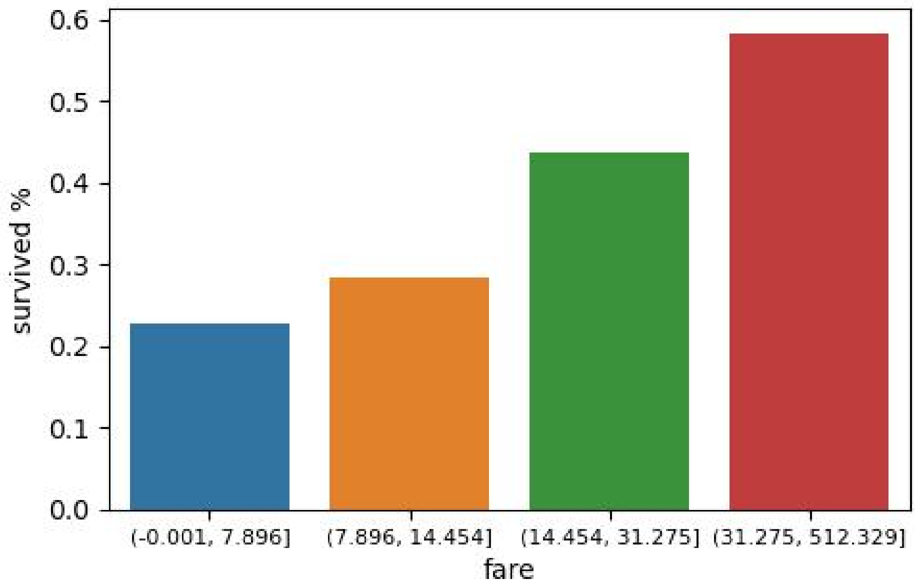
Jak możemy zaobserwować powyżej, zmienne `age` i `fare` mają różnie wyglądające rozkłady wartości. Jeśli chodzi o wiek ludzi, zdecydowanie najwięcej pasażerów było w wieku 16-40 lat, a ogólniej rozkład wartości przypomina rozkład normalny ze średnią w okolicach 30 lat. Natomiast w przypadku zmiennej `fare`, czyli opłacie za bilet, zdecydowanie najwięcej wartości jest z zakresu 0-50, a rozkład wartości swoim kształtem przypomina nieco rozkład eksponencjalny.

Poniżej zaproponowane zostało porównanie wartości zmiennej objaśnianej `survived` w zależności od zmiennej numerycznej `fare`. W tym celu podzieliliśmy wartości tej zmiennej na 4 przedziały, tak aby w każdym z nich znajdowało się tyle samo wartości z naszego zbioru danych. W tym celu użyliśmy metody `qcut` [3] z pakietu **pandas**:

```
df_titanic['fare_przedzialy'] = pd.qcut(df_titanic['fare'], 4)
```

Kod źródłowy 6: Fare przedziały

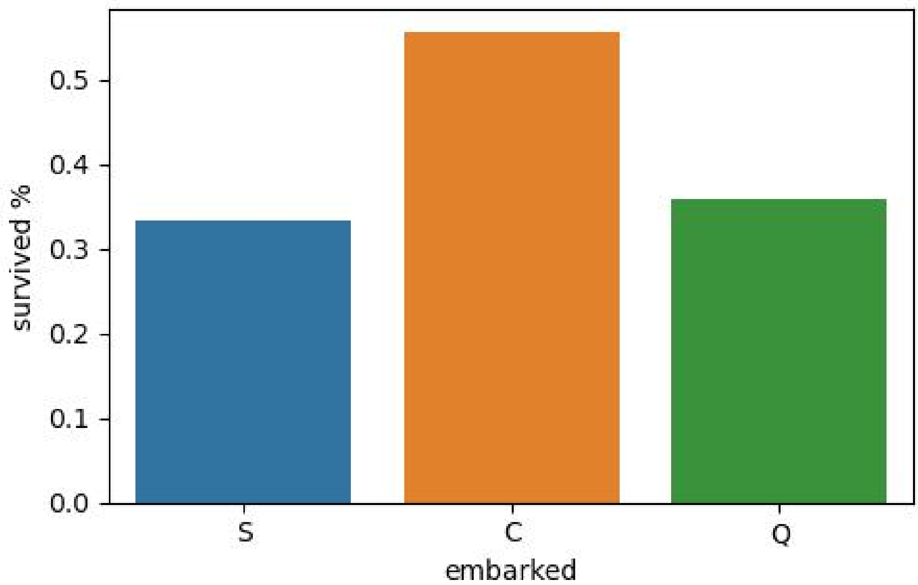
Poniżej, na rysunku 3 zilustrowane zostało porównanie odsetku ocalałych ludzi dla zmiennej `fare`, podzielonej na 4 przedziały.




Rysunek 3: Wskaźniki przeżywalności dla poszczególnych przedziałów zmiennej fare (ceny biletu

Wyraźnie widzimy z powyższego wykresu następującą tendencję: za większymi opłatami za bilety idzie większa średnia wartości zmiennej `survived`, czyli po prostu większy odsetek ocalałych pasażerów. Możemy uzasadnić to tym, że większe ceny biletów przeważnie były równoważne z miejscami w wyższych klasach. Dla przedziału z największymi wartościami zmiennej `fare` odsetek ten jest równy prawie 60%. Kolejne grupy zmiennej `fare`, zaczynając od największych wartości mają odpowiednio 45%, 30% i 25%.

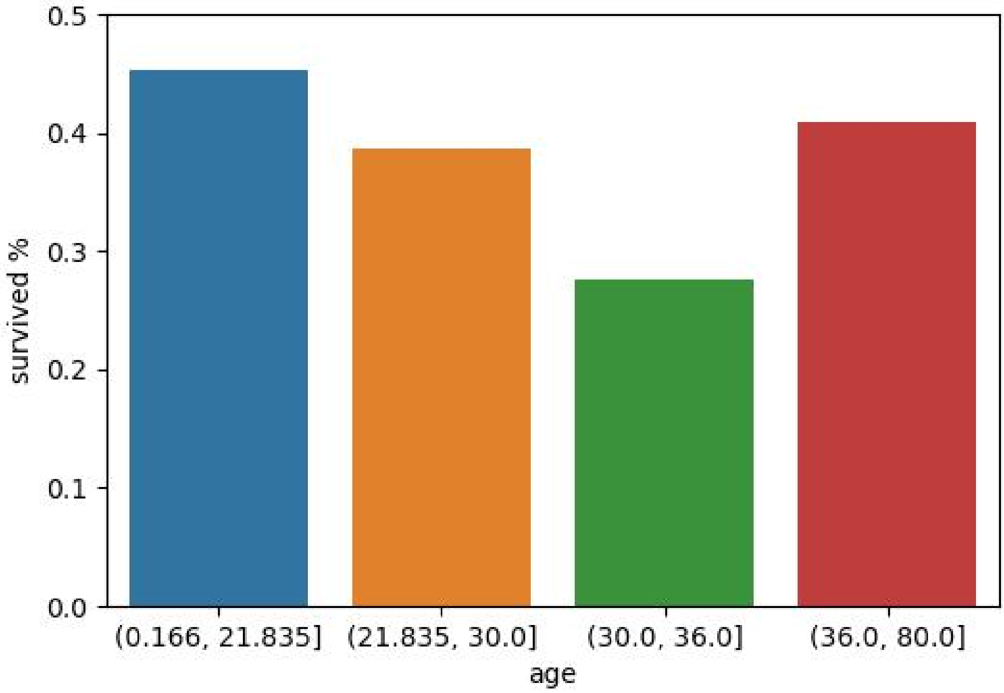
Poniżej, na rysunku 4 przedstawione zostało porównanie średnich wartości zmiennej `survived` w zależności od zmiennej jakościowej `embarked` informującej o porcie w którym wsiadł każdy pasażer.




Rysunek 4: Wskaźniki przeżywalności dla poszczególnych kategorii zmiennej `embarked`

Przypomnijmy, `embarked`=**S** oznacza Souhampton, czyli angielski port z którego wyruszył titanic. Następnie miał przystanek we francuskim porcie Cherbourg (`embarked`=**C**), aby potem skierować się do irlandzkiego miasta nadmorskiego Queenstown (`embarked`=**Q**), dzisiejszego Cobh, skąd wypłynął dalej na zachód w kierunku Nowego Jorku, który był końcowym celem podróży transatlantyku. Jak możemy zauważyć, jeśli chodzi o pasażerów, którzy wsiedli w Souhampton i Queenstown, współczyniki przezywalności są bardzo zbliżone do siebie i są równe w przybliżeniu 35%. Ciekawy jest fakt, że odsetek pasażerów którzy przeżyli, z tych którzy wsiedli we francuskim Cherbourg jest znacząco większy - ponad połowa takich osób przeżyła katastrofę. Nie wiemy do końca czym to jest spowodowane, być może ludzie, którzy weszli na pokład we Francji, byli statystycznie wyższej klasy społecznej, przez co w jakiś sposób bardziej uprzywilejowani przy ewakuacji z podkładu tonącego statku. Niezależnie od powodów ciężko uznać różnicę między portem francuskim, a pozostałymi dwoma za kompletnie nieistotną.

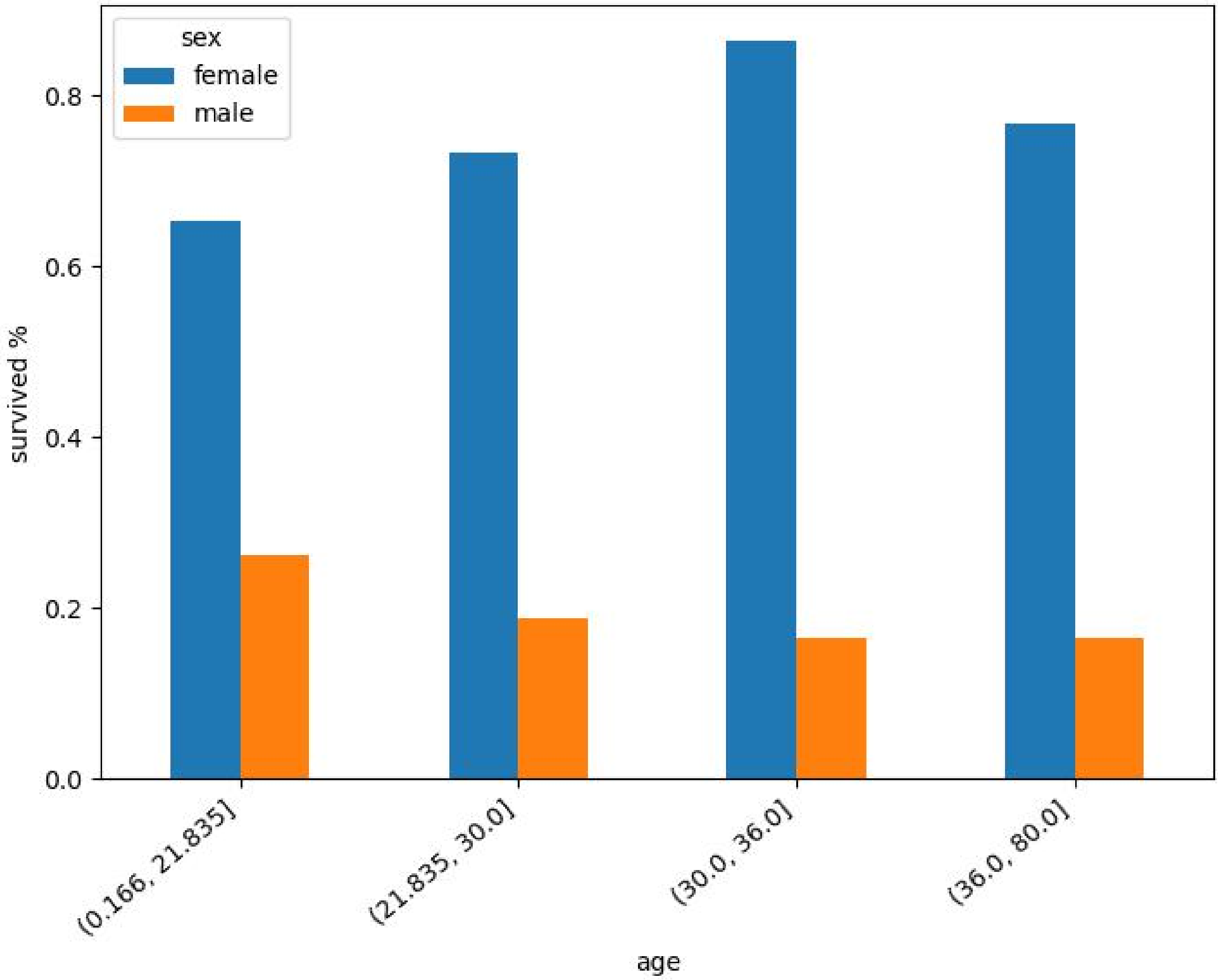
Natomiast na rysunku 5 poniżej przedstawiony jest wykres słupkowy ilustrujący jaki procent ludzi przetrwało katastrofę statku, w zależności od wieku danej osoby. I znowu, dzielimy zmienną numeryczną `age` na 4 grupy (przedziały) za pomocą metody `qcut()`, tak aby do każdej z nich należało tyle samo obserwacji, podobnie jak wcześniej zrobiliśmy to ze zmienną `fare`. Następnie obliczamy średnią zmiennej `survived` dla każdej z 4 grup.




Rysunek 5: Histogramy dla zmiennych ilościowych

Z powyższego rysunku wynika, że nie można określić jednoznacznej tendencji między wiekiem, a odsetkiem ludzi, którzy przeżyli wypadek statku. Widzimy jednak, że największą średnią wartość zmiennej `survived` mają dzieci i młodzież - przedział wiekowy od 0 do 22 lat oraz najstarsza grupa w przedziale wiekowym od 36 do 80 lat. Jest to dość zrozumiałe, ponieważ podczas akcji ratunkowej to właśnie dzieci oraz osoby starsze miały większy priorytet do szalup ratunkowych. Widzimy, że trzecią w kolejności grupą jest przedział między 22, a 30 rokiem życia, i różnice między tymi 3 grupami nie są znaczące - wartości te są w przybliżeniu między 38%, a 45%. Nieco większa różnica jest między wspomnianymi przedziałami wiekowymi, a ostatnim - między 30, a 36 rokiem życia, dla którego odsetek uratowanych jest na poziomie mniej więcej 27%, czyli w dalszym ciągu nie jest to kolosalna różnica między tym, a pozostałymi przedziałami wiekowymi.

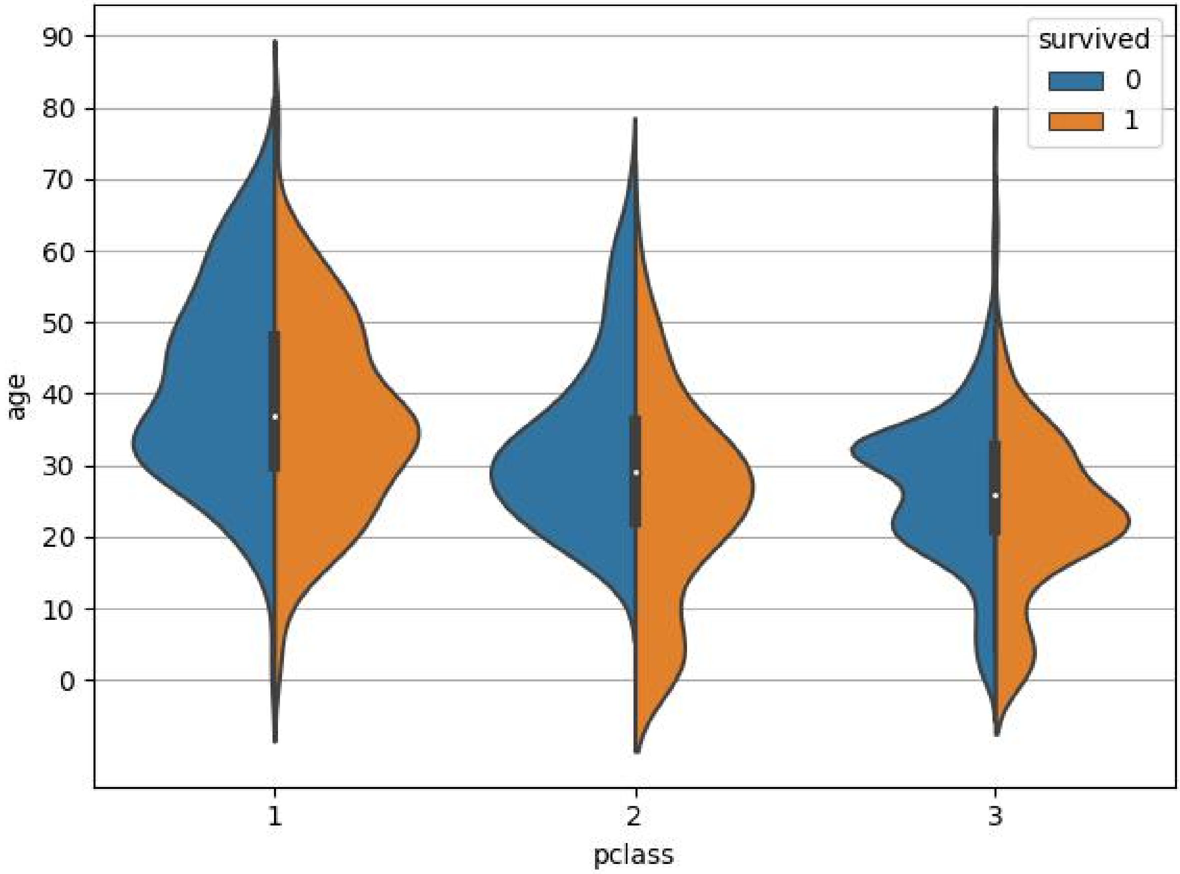
Być może więcej powie nam wykres, w którym oprócz wieku, jednocześnie weźmiemy pod uwagę również płeć pasażera. Każdą z 4 grup wiekowych podzielimy na 2 podgrupy wg. płci, a następnie obliczymy jaki odsetek tych pasażerów się uratowało. Wyniki opisanych wyżej operacji przedstawione zostały na rysunku 6.




Rysunek 6: Częstotliwości występowania zmiennych jakościowych

Z powyższego rysunku jasno wynika, że niezależnie od grupy wiekowej kobiety miały dużo większą przeżywalność, co jest logiczne, bo kobiety miały pierwszeństwo przy umieszczaniu ludzi na łodzie ratunkowe podczas ucieczki z tonącego statku. Najmniejszą dysproporcję między płciami możemy odnotować najmłodszej grupie wiekowej, co może być wynikiem tego, że oprócz kobiet, pierwszeństwo w ratowaniu miały dzieci, w przypadku których płeć była najmniej istotna ze wszystkich grup wiekowych. Jednakże wciąż jest to ponad 2 większy odsetek uratowanych kobiet niż mężczyzn. Skupiając się jedynie na płci męskiej, widzimy jasno tendencję, że wraz ze wzrostem wieku spada odsetek uratowanych. Natomiast sytuacja wygląda dużo ciekawiej w przypadku kobiet, gdzie w najmłodszej grupie było najmniej uratowanych osób, a największy odsetek kobiet, które przeżyły jest w grupie wiekowej 30-36 lat. Takie wartości mogą być spowodowane tym, że jak już wspomnieliśmy wcześniej, wszystkie kobiety, niezależnie od wieku, miały pierwszeństwo przy ewakuacji, a różnice przypuszczalnie mogą wynikać np. z siły danej osoby czy przytomności umysłu, co trochę faworyzuje kobiety w sile wieku.

Kolejnym pomysłem na wizualizacje danych będzie wykres skrzypcowy [4]. Jest to ciekawy, ale rzadko spotykany rodzaj wykresu, który pozwala zilustrować porównanie rozkładów zmiennej ilościowej, przy podziale na kilka grup (np. dwie zmienne jakościowe) jednocześnie. W naszym przypadku zobrazujemy rozkład zmiennej numerycznej `age` przy podziale na kategorie zmiennej `pclass`, zaznaczone na osi poziomej. Dodatkowo, każda kategoria zmiennej `pclass` będzie dodatkowo podzielona na grupy, ze względu na wartości zmiennej `survived`. Wykres ten wykonany został przy pomocy funkcji `violinplot` [5] z biblioteki **seaborn**, a wyniki przedstawione są poniżej na rysunku 7.




Rysunek 7: Histogramy dla zmiennych ilościowych

Z powyższego wykresu możemy wyciągnąć kilka wniosków:

- Jedyne dzieci do lat 10, które nie przeżyły katastrofy miały bilety trzeciej klasy.
- Zauważyliśmy wcześniej z rysunku 2, że dla całego zbioru danych, większość wartości zmiennej `age` znajduje się w przedziale 16-40 lat. Widzimy, że dla biletów drugiej i trzeciej klasy jest podobnie, jednakże dla biletów pierwszej klasy, większość ludzi jest w przedziale 25-60 lat.

- Ogólnie można zaobserwować tendencję, że rozkłady dla poszczególnych klas, dla wartości `survived`=**1** są bardziej skupione niż dla wartości `survived`=**0** w przypadku mniejszych wartości zmiennej `age`, a mniej skupione w przypadku większych wartości zmiennej `age`, czyli dla starszych ludzi.

Na końcu zajmiemy się ostatnimi zmiennymi objaśniającymi, które nie były dotychczas przez nas wykorzystane - `parch` i `sibsp`. Przy pomocy metody `crosstab()` [6] z pakietu **pandas** zostały stworzone dwie tabele krzyżowe. Jedna dla pary `parch` i `survived`, a druga dla pary `sibsp` i `survived`. Wyniki zostały przedstawione odpowiednio w tabelach 4 i 5.

Tabela 4: Tabela krzyżowa dla zmiennych survived i parch

| parch | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 9 |
|---|---|---|---|---|---|---|---|---|
| survived | | | | | | | | |
| 0 | 666 | 70 | 56 | 3 | 5 | 5 | 2 | 2 |
| 1 | 336 | 100 | 57 | 5 | 1 | 1 | 0 | 0 |

Tabela 5: Tabela krzyżowa dla zmiennych survived i sibsp

| sibsp | 0 | 1 | 2 | 3 | 4 | 5 | 8 |
|---|---|---|---|---|---|---|---|
| survived | | | | | | | |
| 0 | 582 | 156 | 23 | 14 | 19 | 6 | 9 |
| 1 | 309 | 163 | 19 | 6 | 3 | 0 | 0 |

Wyniki w przypadku obu zmiennych prezentują się bardzo podobnie. Widzimy, że zarówno w przypadku zmiennej `parch`, jak i `sibsp` dla wartości **0**, ok. 1/3 ludzi przeżyło katastrofę, dla wartości **1** i **2**, ok. połowa ludzi przeżyło, a dla wartości większych niż **2**, odsetek uratowanych ludzi z powrotem spada.

# 2 Metody uczenia zespołowego

## 2.1 Ogólna idea

Uczenie zespołowe (ang. `ensemble learning`) to technika w dziale uczenia maszynowego, która polega na stworzenie wielu pojedynczych, elementarnych modeli (ang. `weak learners)` w celu połączenia ich i stworzenia jednego, końcowego modelu predykcyjnego (ang. `strong learner`), który byłby lepszy. W przypadku klasyfikacji, najpowszechniejszą metodą łączenia pojedynczych klasyfikatorów jest głosowanie, która polega na tym, że każdy podstawowy model osobno przewiduje wartość zmiennej jakościowej, a potem prognozę, która wystąpiła najwięcej razy, traktujemy jako ostateczną predykcję modelu.

## 2.2 Pojedynczy klasyfikator - drzewo decyzyjne

Metody uczenia zespołowego mogą być aplikowane na praktycznie każdym podstawowym klasyfikatorze, jednakże zdecydowanie najpowszechniejsze jest drzewo decyzyjne [7] (ang. `decision tree`), które będzie także naszym wyborem. Najważniejszymi parametrami drzewa decyzyjnego są:

- `max depths` - parametr określający maksymalną głębokość drzewa. Im większa ta wartość, tym bardziej złożone drzewo powstanie. Wartość ta nie może być zbyt mała, ponieważ wtedy drzewo może mieć zbyt małą elastyczność żeby uchwycić wzorce i potencjalne interakcje w zbiorze treningowym. Jednakże jeśli ustawimy zbyt dużą wartość tego parametru, wtedy istnieje ryzyko, że nastąpi przeuczenie modelu do zbioru treningowego, co również będzie wiązało się ze wzrostem błędu na zbiorze testowym.

- `min samples split` - parametr, który określa ile przynajmniej obserwacji musi znajdować się w węźle, aby móc go podzielić.

- `min samples leaf` - minimalna liczba określająca węzeł, którego nie chcemy dalej dzielić (ang. `leaf node`).

## 2.3 Bagging

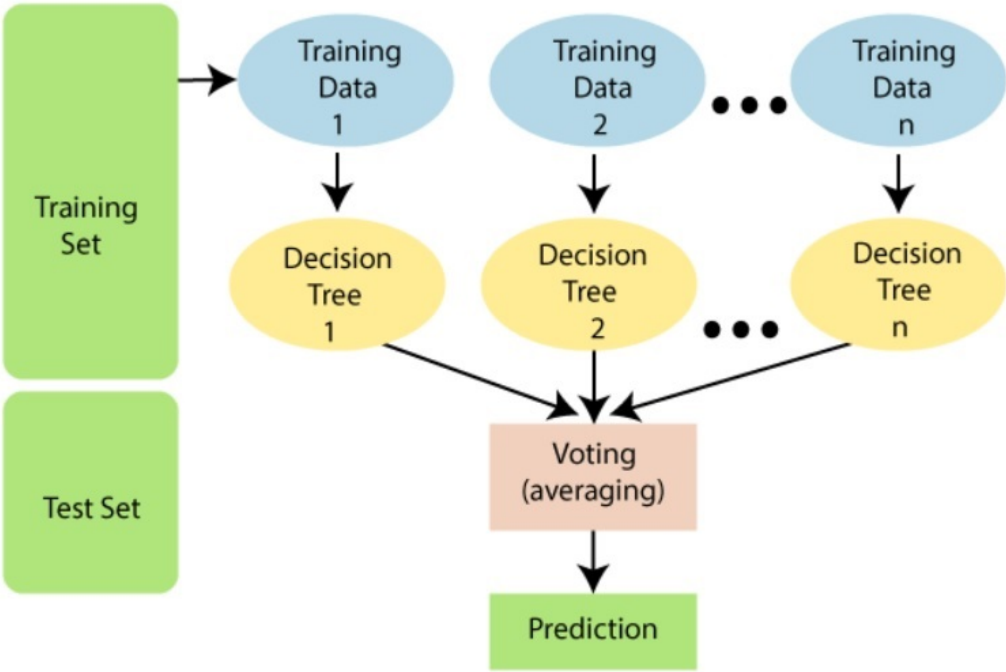
`Bagging` [8] (Bootstrap aggregating), to prawdopodobnie najpopularniejsza technika uczenia zespołowego należąca do grupy równoległych metod zespołowych, które przede wszystkim wykorzystują niezależność między podstawowymi klasyfikatorami, co pozwala zmniejszyć błąd poprzez uśrednienie. Warto dodać, że w przypadku, gdy podstawowym klasyfikatorem jest drzewo decyzyjne, to w przypadku `bagging'u` często mówimy o tzw. `bagged trees`.

Na początku warto wyjaśnić czym jest próba `bootstrap` [9]. Jest to po prostu losowanie ze zwracaniem $n$ obserwacji ze zbioru wszystkich obserwacji, który ma $n$ elementów. Innymi słowy obserwacje mogą się powtarzać w próbce, ale także możliwe (wręcz prawie pewne) jest, że niektóre obserwacje zostaną pominięte. Oczywiście parametrem, który przeważnie możemy modyfikować jest $B$ - liczba boostrapowych prób, które chcemy losować.

Dysponując $B$ bootstrapowymi próbami, dla każdej z nich dopasowujemy nasze drzewo decyzyjne, a następnie dysponując $B$ drzewami, możemy interpretować kategorię zwracaną przez każde z nich jak "oddany głos", a następnie kategoria która otrzyma najwięcej (tzw. `soft-voting`) albo większość, czyli ponad połowę (tzw. `hard-voting`) głosów jest zwracana przez model zespołowy jako ostateczna prognoza.

## 2.4 Random forest

`Random forest` (las losowy) [10] to szczególny przykład metody `bagged trees`. Polega na tym samym co metoda `bagged trees`, ale dodatkowo przy każdym losowaniu próby bootstrap, losowany jest także podzbiór cech wykorzystywanych przy tworzeniu danego pojedynczego drzewa decyzyjnego. Ile zmiennych jest losowanych? Otóż jeśli $p$ to liczba wszystkich zmiennych, to oczywiście można wybrać jakąkolwiek liczbę mniejszą od $p$, jednakże przyjmuje się, że odpowiednimi wartościami są $\sqrt{p}$ dla problemu klasyfikacji oraz $p/3$ dla regresji. Na rysunku 8 pokazany jest zwizualizowany schemat działania techniki `random forest`.



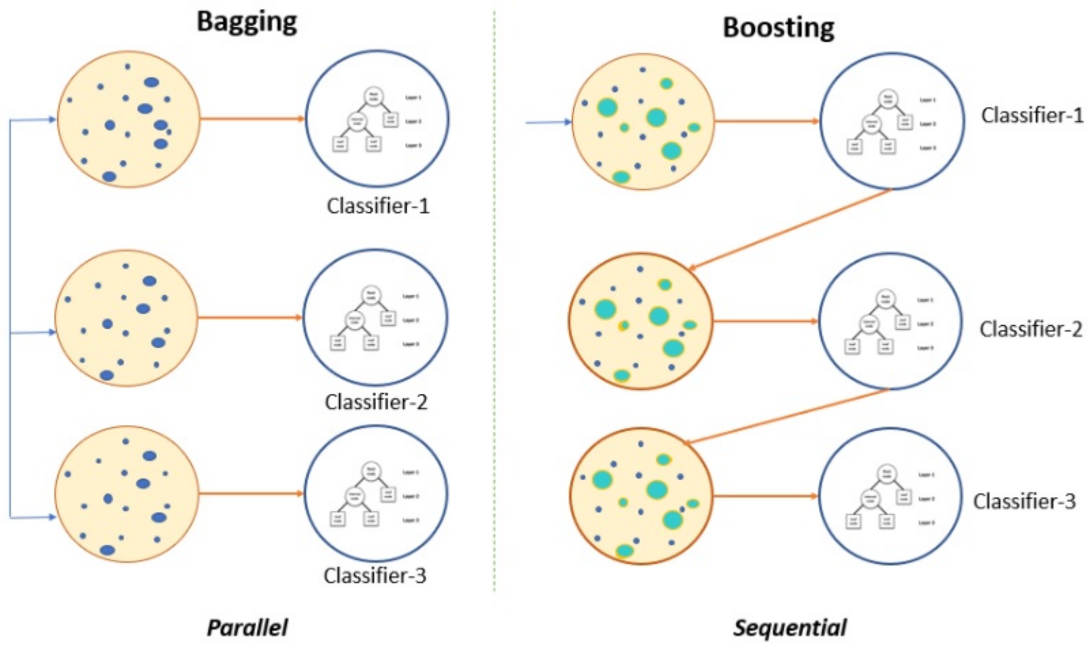

Rysunek 8: Schemat działania metody `random forest`

## 2.5 Boosting

`Boosting` (wzmacnianie) [11] to rodzina algorytmów w uczeniu maszynowym, które konwertują klasyfikatory słabe (ang. `weak learners`) w klasyfiktory mocne (ang. `strong learners`). Przez kolejne iteracje, algorytm stara się wykorzystywać informacje o podstawowym klasyfikatorze przy tworzeniu końcowego klasyfikatora. `Boosting` może być postrzegane jako uogólnienie metody `bagging`.

Większość algorytmów wzmacniania polega na iteracyjnym uczeniu się słabych klasyfikatorów z uwzględnieniem rozkładu i dodawaniu ich do ostatecznego silnego klasyfikatora. Po dodaniu słabego klasyfikatora, wagi poszczególnych obserwacji z naszych danych są ponownie dostosowywane, proces ten nazywamy ponownym ważeniem (tzw. `resampling` [12]). Błędnie sklasyfikowane dane wejściowe zyskują większą wagę, a przykłady, które zostały poprawnie sklasyfikowane, tracą na wadze. Tak więc przyszłe słabe klasyfikatory skupiają się bardziej (tzn. jest większe prawdopodobieństwo, że zostaną wylosowane do próby bootstrapowej), na tych obserwacjach, które zostały przez poprzednie słabe klasyfikatory błędnie sklasyfikowane.

Poniżej na rysunku 9 przedstawione zostało porównanie przedstawiające różnice między metodami `bagging` i `boosting`. Warto zwrócić uwagę na fakt, że `bagging` jest przykładem równoległej metody zespołowej, natomiast `boosting` jest metodą sekwencyjną.




Rysunek 9: Różnica działania metod `bagging` i `boosting`

Pierwszym głośnym algorytmem `boosting`'u był **AdaBoost** [13] zaproponowany przez Roberta Shapire'a i Yoav Freunda w 1995r. [14] Algorytm ten był przeznaczony tylko i wyłącznie do klasyfikacji binarnej. Rozszerzeniem tego algorytmu na problemy klasyfikacji wieloklasowej jest algorytm **AdaBoost.M1** , zaproponowany przez Roberta Shapire'a i Yoav Freunda w 1996r. [15] Poniżej przedstawiony jest schemat działania algorytmu AdaBoost.M1:

1. Ustal początkowe wagi przypadków $\mathbf{w}^{(1)} = \left(w_1^{(1)}, w_2^{(1)}, ..., w_n^{(1)}\right)'$, gdzie $w_j^{(1)} \in [0, 1]$ oraz $\sum_{i=1}^{n} w_i^{(1)} = 1$. Zazwyczaj przyjmuje się, że $w_j^{(1)} = 1/n$.
2. Dla $b = 1, 2, ..., B$ :
   (a) wyznacz próbę bootstrapową $\mathcal{L}_n^{*b}$ na bazie $\mathcal{L}_n$, przyjmując $P((x_j, g_j) \in \mathcal{L}_n^{*b}) = w_j^{(b)}$ ($x_j$ -zmienne objaśniające, $g_j$ - zmienna objaśniana),
   (b) skonstruuj klasyfikator $\hat{d}_b$ na podstawie próby bootstrapowej$\mathcal{L}_n^{*b}$,
   (c) wyznacz ważony błąd klasyfikacji
   $\hat{e}_b = \sum_{i=1}^{n} w_i^{(b)} I_i^{(b)}$, gdzie $I_i^{(b)} = \begin{cases} 1 & \hat{d}_b(x_i) \neq g_i \\ 0 & \hat{d}_b(x_i) = g_i \end{cases}$
   (d) jeżeli $\hat{e}_b \in (0, 1/2)$ oblicz $\beta_b = \frac{\hat{e}_b}{1-\hat{e}_b}$. W przeciwnym wypadku przyjmij $w_i^{(b)} = 1/n$ i wróć do kroku a),
   (e) aktualizuj wagi $w_j^{(b+1)} = \frac{w_j^{(b)} \beta_b^{1-I_j^{(b)}}}{\sum_{i=1}^{n} w_i^{(b)1-I_j^{(b)}}}$, $j = 1, 2, ..., n$.
3. Wyznacz końcowy klasyfikator

$$\hat{d}_{AdaBoost}(\mathbf{x}) = \underset{1 \le k \le K}{\text{argmax}} \sum_{b=1}^{B} \left[ ln\left(\frac{1}{\beta_b}\right) \mathbb{1}\left(\hat{d}_b(\mathbf{x}) = k\right)\right]$$

Jak możemy zauważyć powyżej, poprzez iteracyjne aktualizowanie wag, w odróżnieniu od metody `bagging`, klasyfikatory składowe w **AdaBoost** nie są tworzone niezależnie od siebie: aktualny klasyfikator zależy od klasyfikatorów w poprzednich krokach. Widzimy też, że klasyfikatory składowe otrzymują wagi równe $ln\left(\frac{1}{\beta_b}\right)$, tzn. przy konstrukcji klasyfikatora złożonego za pomocą głosowania dajemy większą wagę dokładniejszym klasyfikatorom. Przypadki $\{w_i^{(b)}\}_{i=1,\ldots,n}$ uwzględniane są w b-tej iteracji za pomocą losowania $n$ obserwacji ze zwracaniem, z prawdopodobieństwem proporcjonalnym do ich wag, czyli jest to przykład ponownego ważenia.

# 3 Klasyfikacja zmiennej objaśnianej

## 3.1 Opis symulacji

Na początku wybrane zostało 9 wariantów opisanych wyżej metod:

- pojedyncze drzewo decyzyjne
- `bagged trees` z 25 bootstrapowymi replikacjami
- `bagged trees` ze 150 bootstrapowymi replikacjami
- `random forest` z 25 bootstrapowymi replikacjami oraz podzbiorem 3 losowych cech
- `random forest` ze 150 bootstrapowymi replikacjami oraz podzbiorem 3 losowych cech
- `boosting` z 25 iteracjami
- `boosting` ze 150 iteracjami
- `boosting` z 25 iteracjami oraz aktualizacją wag zaproponowaną przez Leo Breimana
- `boosting` ze 150 iteracjami oraz aktualizacją wag zaproponowaną przez Leo Breimana

Dodajmy, że w ostatnich dwóch metodach wagi zaproponowane przez Breimana różnią nieznacznie (przemnożone o stałą) względem tego co zaproponował Freund. Więcej o tych różnicach można przeczytać tutaj [16].

Symulacja polegała na obliczeniu dokładności, czułości oraz swoistości [17] każdej z wyżej wymienionych metod. Losowany był zbiór uczący o wielkości 70% wszystkich obserwacji, a reszta trafiła do zbioru testowego. Na podstawie zbioru uczącego skonstruowane zostały klasyfikatory złożone (oraz prosty w przypadku drzewa decyzyjnego), a następnie na zbiorze testowym obliczyliśmy dokładność, czułość oraz swoistość. Wszystkie te operacje powtórzyliśmy 100 razy, oczywiście gromadząc wyniki.

## 3.2 Sposoby implementacji

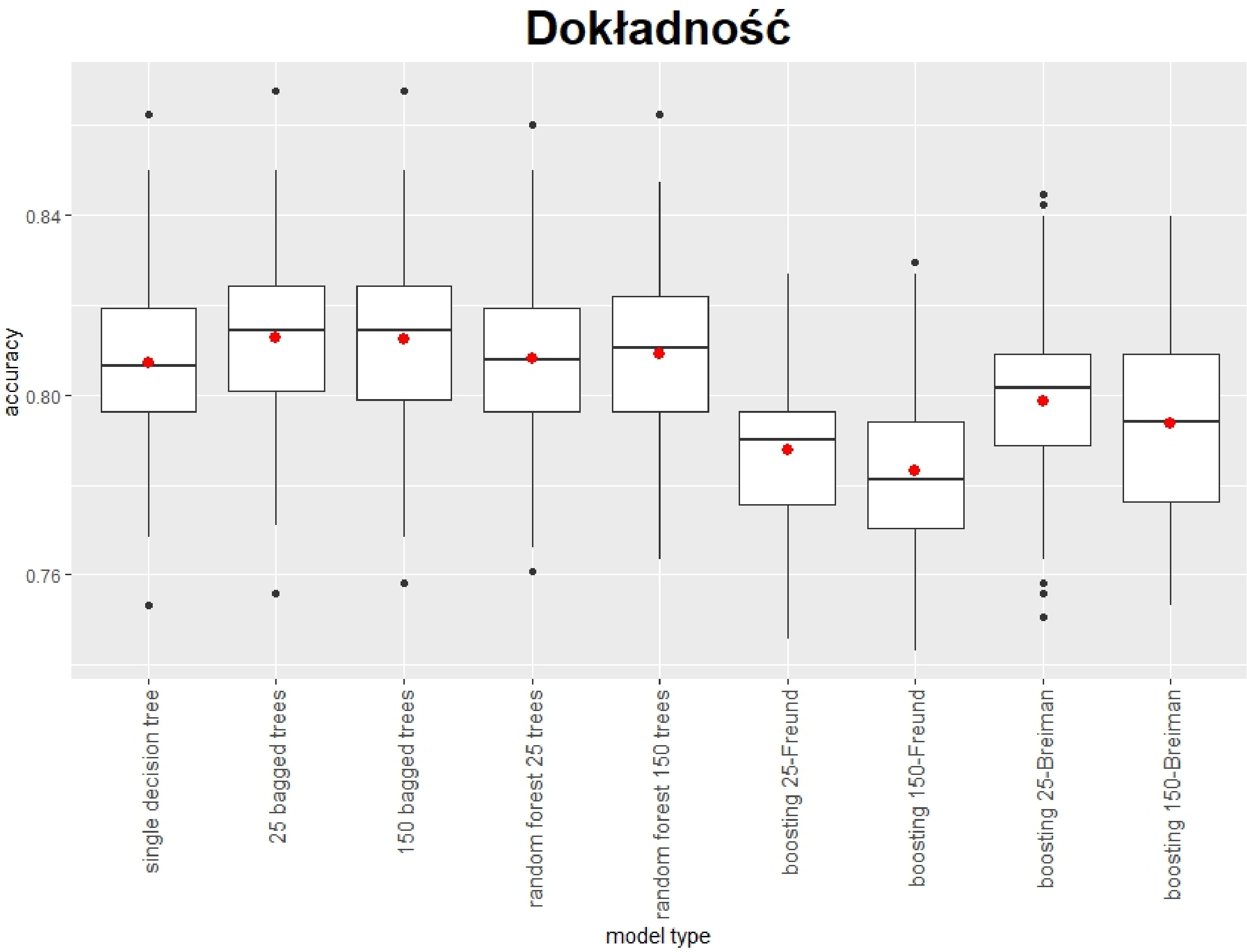
Cała symulacja została przeprowadzona w **R**, ponieważ w języku **Python** w pakiecie **scikit-learn** nie istnieje sposób, żeby uwzględnić zmienne jakościowe bez potrzeby kodowania.

Podstawowy klasyfikator, czyli drzewo decyzyjne zostało zaimplementowane za pomocą funkcji `rpart()` z pakietu **rpart**, z ustalonym parametrem `minsplit`$= 2$. Wartość ta została wybrana specjalnie tak mała, żeby drzewo było bardziej złożone i potencjalnie zwiększyć niestabilność tej metody. Metoda `bagged trees` została zaimplementowana za pomocą funkcji `bagging()` z pakietu **ipred**, również z ustalonym parametrem `minsplit`= 2, tylko tym razem za pomocą funkcji `rpart.control()` . Metoda `random forest` została wdrożona za pomocą funkcji `randomForest()` z pakietu **randomForest**. Ustawiliśmy parametr kontrolujący minimalną liczbę węzłów końcowych `nodesize`= 5. Metoda `boosting` została zaimplementowana za pomocą funkcji `boosting()` z pakietu **adabag**. Funkcja ta używa algorytmu **AdaBoost.M1** z wykorzystaniem drzewa decyzyjnego jako pojedynczego klasyfikatora do obliczenia złożonego klasyfikatora. I tutaj również ustaliliśmy paramter `minsplit`= 2.

## 3.3 Wyniki

Tak jak wspomnieliśmy wcześniej, zostało zebranych po 100 wartości liczbowych dokładności, czułości i swoistości dla każdej z 9 metod, które były rozpatrywane.

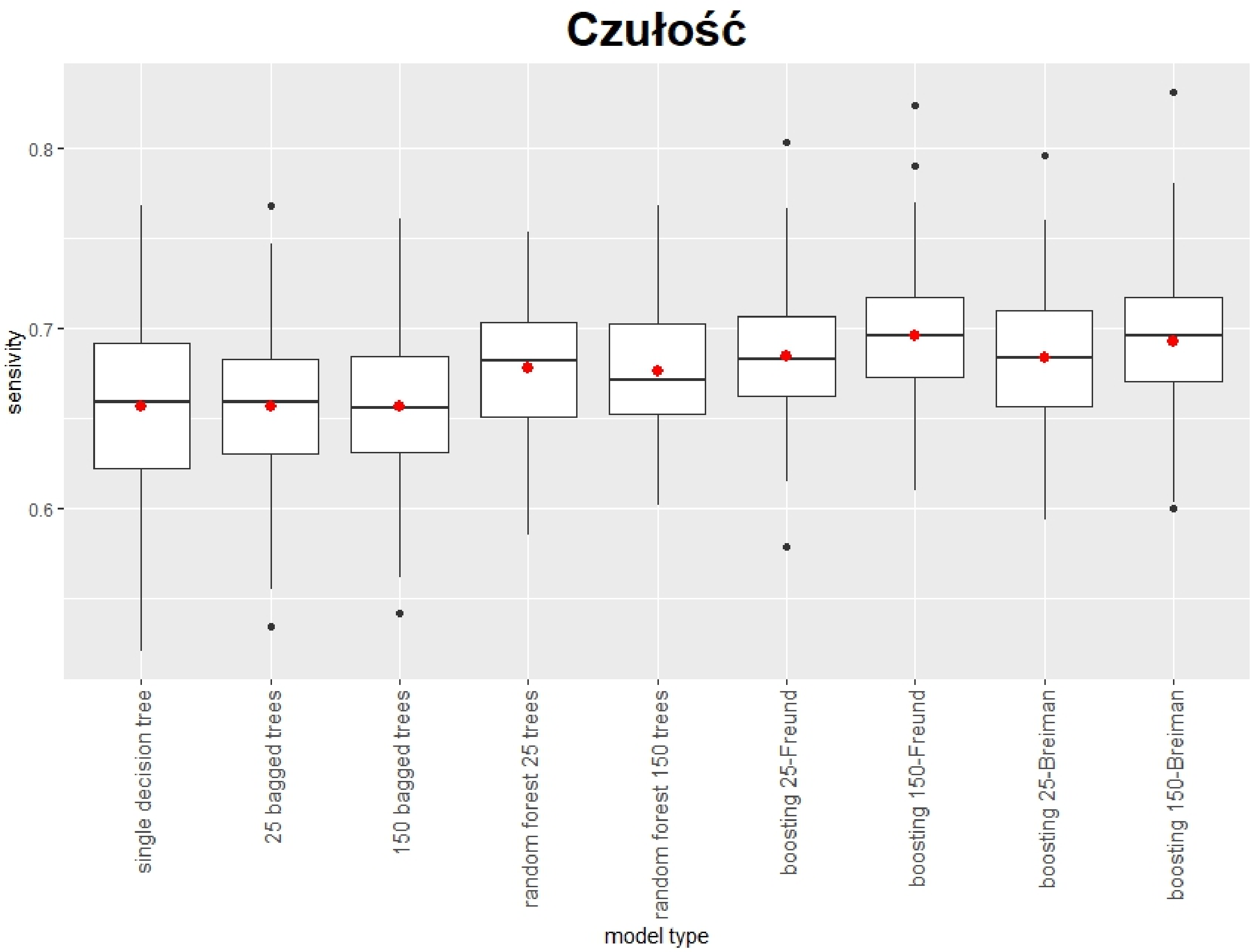
Na początek zajmiemy się dokładnością, która jest prawdopodobnie najpowszechniejszą i najwięcej mówiącą miarą oceny klasyfikacji. Jest to stosunek poprawnie zaklasyfikowanych obiektów do wszystkich obserwacji. Na rysunku 10 zaprezentowane zostały wykresy pudełkowe przedstawiające dokładności dla wszystkich opracowanych metod. Do wykresów zostały dodane wartości średnie dla każdej metody, które zostały oznaczone czerwonymi kropkami.




Rysunek 10: Porównanie dokładności drzewa decyzyjnego oraz metod uczenia zespołowego dla zbioru danych Titanic

Najważniejszą rzeczą jaką widzimy z powyższego wykresu widzimy jest to, że żadna z metod uczenia zespołowego zaimplementowana na naszym zbiorze danych nie poprawiła znacząco dokładności względem podstawowego modelu - drzewa decyzyjnego, którego średnia dokładność to ok. 0.81. Przyczyn tego może być kilka, ale najbardziej prawdopodobna jest taka, że nasz zbiór danych jest zbyt mało skomplikowany, tym samym klasyfikacja zmiennej `survived` to **problem o niewystarczającej złożoności**. Rozumiemy przez to, że problem jest stosunkowo zbyt prosty, przez co pojedyncze drzewo klasyfikacyjne jest w stanie na tyle dobrze go uchwycić, że wprowadzenie modeli zespołowych nie przynosi poprawy. Co więcej, widzimy, że wszystkie 4 warianty `boosting`'u nawet pogorszyły średnią dokładność. Najlepsze wyniki jeśli chodzi o dokładność, ale tylko nieznacznie, uzyskała metoda `bagged trees`, ale również jest to średnio tylko ok. 0.82, a najgorzej poradziły sobie warianty `boosting`'u z aktualizacjami wag od Freunda ze średnimi na poziomie ok. 0.785. Widzimy, że w przypadku `bagging`'u i `random forest` liczba bootstrapowych próbek nie miała dużego znaczenia dla ewentualnej poprawy rezultatów, natomiast w przypadku obu wariantów `boosting`'u większa liczba iteracji miała nawet negatywny wpływ na średnią dokładność. Jeśli chodzi o rozrzuty wyników, to nie obserwujemy większych różnic, poza `boosting`'iem ze 150 iteracjami z aktualizacją wag Breimana, gdzie wyniki mają dużo większy rozstęp międzykwartylowy.

Na rysunku 11 poniżej przedstawione zostały wykresy pudełkowe dla czułości. Dodano także wartości średnie dla każdej metody, oznaczone czerwonymi kropkami. Przypomnijmy, że czułość (ang. `sensitivity`) to stosunek poprawnie zaklasyfikowanych pozytywnych przypadków do wszystkich rzeczywistych pozytywnych przypadków. Jednakże w klasycznych testach np. w medycynie czy w finansach "pozytywny" lub "dodatni" wynik testu oznacza coś negatywnego np. występowanie choroby u pacjenta czy niewypłacalność klienta, a w naszym przypadku "pozytywny" przypadek (`survived`= 1) oznacza coś pozytywnego w istocie, bo przeżycie katastrofy przez daną osobę.

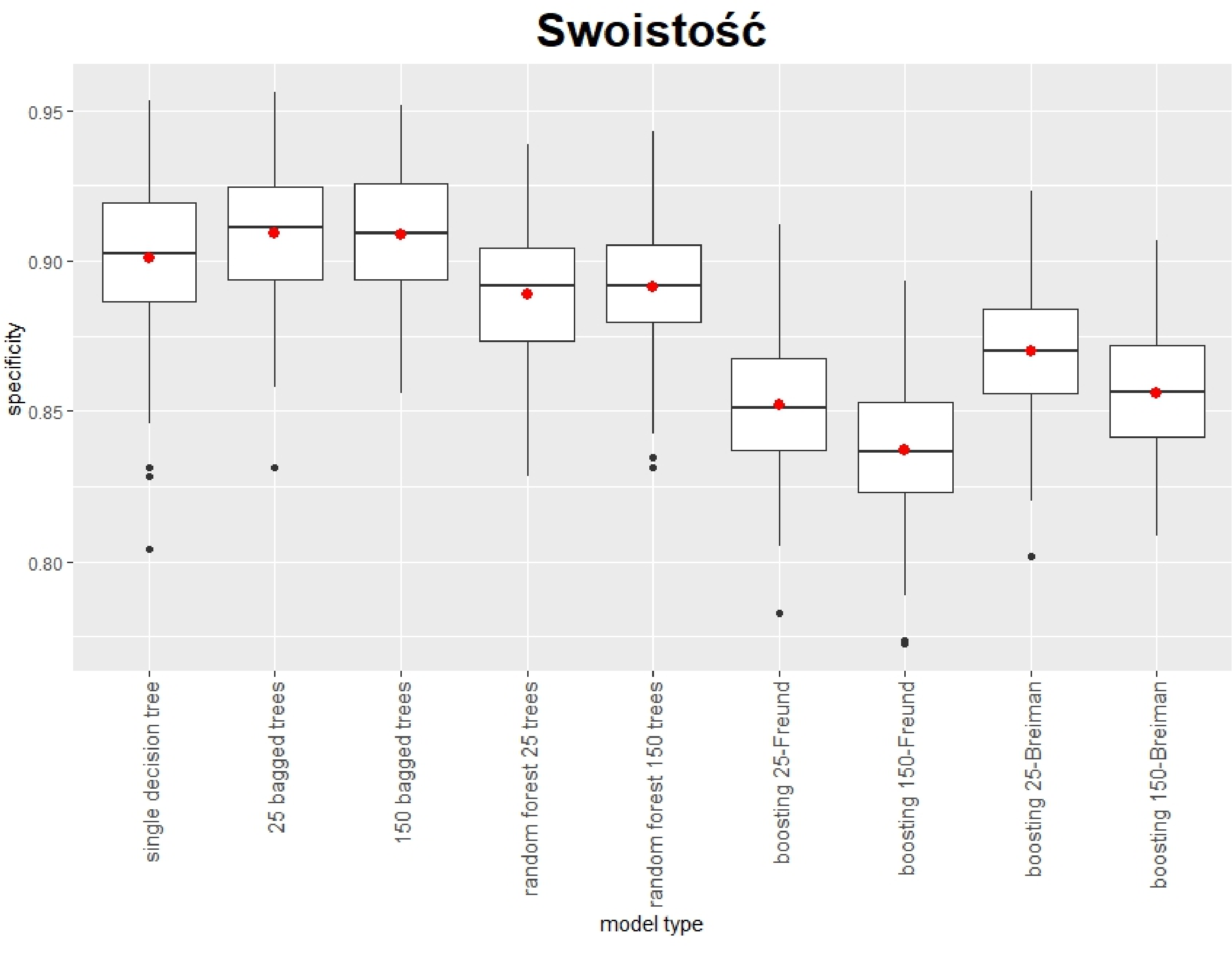



Rysunek 11: Porównanie czułości drzewa decyzyjnego oraz metod uczenia zespołowego dla zbioru danych Titanic

Jak obserwujemy powyżej, jeśli chodzi o czułość to wartość średnia równa w przybliżeniu

0.61 dla drzewa decyzyjnego jest najmniejsza i reszta metod ma średnie czułości na podobnym poziomie (`bagged trees`) lub na lepszym (`random forest` - ok. 0.66 i `boosting` 0.67-0.7). Najlepsze wyniki czułości uzyskały oba warianty `boosting`'u ze 150 iteracjami. Widzimy, że w przypadku metod `bagged trees` i `random forest` liczba próbek bootstrapowych nie miała znaczenia dla średnich wartości czułości, natomiast w przypadku `boosting`'u wraz z większą liczbą iteracji wystąpił nieznaczny wzrost średniej. Ponadto, rodzaj aktualizacji wag w `boosting`'u nie miał praktycznie żadnego znaczenia dla wartości czułości. Patrząc na rozrzuty rezultatów, możemy stwierdzić, że zdecydowanie największy rozstęp międzykwartylowy miało drzewo decyzyjne.

Rysunek 12 poniżej przedstawia wykresy pudełkowe swoistości, czerwonymi kropkami zaznaczono wartości średnie dla odopowiednich wariantów metod. Warto przypomnieć, że swoistość (ang. `specificity`) to stosunek obserwacji poprawnie zaklasyfikowanych jako negatywne do wszystkich obserwacji, które faktycznie były negatywne (`survived`= 0).




Rysunek 12: Porównanie swoistości drzewa decyzyjnego oraz metod uczenia zespołowego dla zbioru danych Titanic

Jak możemy zaobserwować powyżej, wartości swoistości są dużo większe od czułości, co mówi nam, że wszystkie zaproponowane modele dużo lepiej poradziły sobie z wykryciem kategorii `survived`= 0, czyli pasażerów, którzy nie przeżyli katastrofy. Widzimy, że drzewo decyzyjne uzyskało bardzo dobry średni wynik ok. 0.9 i tylko oba modele `bagged trees` uzyskały lepszy średni rezultat na poziomie ok. 0.91. Najgorzej natomiast poradziły sobie algorytmy `boosting`'u, w szczególności te z aktualizacją wag Freunda z najmniejszym średnim wynikiem ok. 0.83. Ponadto widzimy, że wyniki dla `boosting`'u są gorsze dla większej liczby iteracji algorytmu.

# 4 Podsumowanie

Głównym celem powyższej pracy było przedstawienie najpopularniejszych metod uczenia zespołowego, implementacja ich na wybranym przez nas zbiorze danych Titanic, a następnie sprawdzenie jaki wpływ będą mieć na wyniki klasyfikacji. Zanim to jednak nastąpiło, w rozdziale 1 przeprowadziliśmy analizę eksploracyjną obejmującą przygotowanie danych do analizy - czyli potrzebne transformacje - oraz analizę poszczególnych zmiennych, zarówno wizualną, jak i formalną. Następnie opisaliśmy krótko wybrane przez nas techniki uczenia zespołowego, a na końcu przeprowadziliśmy symulacje w której sprawdziliśmy ich wpływ na dokładność klasyfikacji zmiennej objaśnianej `survived`.

Po wykonaniu symulacji i analizie wyników okazało się, że metody uczenia zespołowego nie przyniosły praktycznie żadnego efektu jeśli chodzi o poprawę dokładności. Niektóre z nich zwróciły rezultaty bardzo przybliżone, a niektóre z nich (`boosting`) nawet gorsze. Przypuszczalnie wybrany zbiór danych w naszym przypadku był zbyt mało skomplikowany, co sprawiło, że klasyfikacja była na tyle prosta, że pojedynczy podstawowy klasyfikator - czyli drzewo decyzyjne - był modelem wystarczającym, aby odpowiednio uchwycić ten problem. Być może wystąpiła także niewystarczająca różnorodność w bazowych modelach. W przypadku, gdy modele składowe są zbyt podobne i mają podobne wady, połączenie ich nie musi prowadzić nas do poprawy dokładności. Przyczyn tego, dlaczego w naszym przypadku metody zespołowe nie przyniosły pożądanego skutku, może być dużo. Warto jednak zwrócić uwagę na fakt, że poradziły one sobie lepiej przy pomiarach czułości (czyli przy wykrywaniu kategorii `survived`$=1$) niż pojedyncze drzewo klasyfikacyjne.

Ostatecznym wnioskiem powinno być to, żeby nie używać metod uczenia zespołowego bez jakiegokolwiek zastanowienia, licząc na poprawę wyników klasyfikacji, ponieważ nie w każdej sytuacji otrzymamy oczekiwane rezultaty. Przede wszystkim, trzeba przeanalizować czy wybrany bazowy klasyfikator nie jest dla nas wystarczający, żeby poradzić sobie z analizowanym problemem i zwrócić zadowalające wyniki. Natomiast jeśli zdecydujemy się, że podstawowy klasyfikator to za mało, to przy dzisiejszej mnogości różnych technik i metod z zakresu uczenia zespołowego, a także tego, ile w każdej z tych metod trzeba dostroić różnych parametrów, należy się zastanowić i spróbować wielu różnych potencjalnych rozwiązań w pogoni za oczekiwaną poprawą wyników.

# Literatura


[1] https://seaborn.pydata.org/generated/seaborn.countplot.html

[2] `https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.hist.html`

[3] https://pandas.pydata.org/docs/reference/api/pandas.qcut.html

[4] `https://en.wikipedia.org/wiki/Violin_plot`

[5] https://seaborn.pydata.org/generated/seaborn.violinplot.html

[6] https://pandas.pydata.org/docs/reference/api/pandas.crosstab.html

[7] https://pl.wikipedia.org/wiki/Drzewo_decyzyjne

[8] https://en.wikipedia.org/wiki/Bootstrap_aggregating

[9] https://pl.wikipedia.org/wiki/Bootstrap_(statystyka)

[10] https://pl.wikipedia.org/wiki/Las_losowy

[11] https://en.wikipedia.org/wiki/Boosting_(machine_learning)

[12] https://en.wikipedia.org/wiki/Sample-rate_conversion

[13] https://en.wikipedia.org/wiki/AdaBoost

[14] Freund, Yoav; Schapire, Robert E. (1995), *A desicion-theoretic generalization of on-line learning and an application to boosting*, Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 23–37

[15] Freund, Y.; Schapire, R.E., et al., (1996), *Experiments with a new boosting algorithm.* In: Icml, Vol. 96. Citeseer, pp. 148–156

[16] Alfaro, E.; Gamez, M., García, N. (2013). *adabag: An R Package for Classification with Boosting and Bagging.* Journal of Statistical Software, 54(2), 1–35

[17] https://en.wikipedia.org/wiki/Sensitivity_and_specificity

[18] https://geek.justjoin.it/ensemble-learning-czym-czym-polega/#Glosowanie_i_usredniani

[19] https://en.wikipedia.org/wiki/Ensemble_learning

[20] https://www.educba.com/decision-tree-hyperparameters/

[21] https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

[22] https://www.javatpoint.com/machihttps://twitter.com/home?lang=plne-learning-random-forest-algorithm

[23] Adam Zagdański, wykłady data mining, Politechnika Wrocławska

[24] https://pl.frwiki.wiki/wiki/Boosting

[25] https://miroslawmamczur.pl/005-wykres-skrzypcowy-violin-plot/