Klasyfikator oparty na twierdzeniu Bayesa przy naiwnym założeniu o wzajemnej niezależności atrybutów

Łukasz Odwrot 218283

13.03.2018

Spis treści

1 Wstęp

Naiwny klasyfikator bayesowski to prosty klasyfikator probabilistyczny oparty o twierdzenie Bayesa i założeniu o niezależności zmiennych losowych. Dla danej klasy obiektu y i wektora cech X na podstawie twierdzenia Bayesa prawdziwy jest wzór:

$$P(y \mid x_1, ..., x_n) = \frac{P(y)P(x_1, ...x_n \mid y)}{P(x_1, ..., x_n)}$$

Korzystając z założenia o niezależności zdarzeń i przekształceń można dojść do wzoru:

$$\hat{y} = arg \max_y P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

Dzięki takiemu mechanizmowi na podstawie ciągu uczącego można wytrenować klasyfikator, a następnie wykorzystać go do klasyfikacji nowych obiektów. Do badania jakości uzyskanych klasyfikatorów użyte zostaną następujące mechanizmy: Confusion Matrix, ccuracy, Precision, Recall, Fscore. Badania zostaną przeprowadzone na trzech zbiorach danych: Iris, Wine oraz Diabetes.

2 Implementacja klasyfikatora i problem wygładzania

Na podstawie zaprezentowanych wcześniej wzorów można stwierdzić, że w przypadku, gdy dana kombinacja cechy i wartości nie wystąpi w zbiorze uczącym, wyzeruje ona prawdopodobieństwo klasyfikacji do danej klasy przy wystąpieniu cechy w czasie klasyfikacji właściwej. Z tym zjawiskiem można poradzić sobie poprzez wygładzenie danych, czyli eliminację zerowych prawdopodobieństw lub założenie, że dane mają rozkład normalny. W tym wypadku można skorzystać ze wzoru, który likwiduje zerowe prawdopodobieństwa.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp(-\frac{(x_i - \mu_y)^2)}{2\sigma_y^2})$$

Do badań wykorzystane zostaną dwie implementacje pochodzące z biblioteki *sklearn*: *GaussianNB* oraz *MultinomialNB* ze współczynnikiem wygładzania $\alpha = 1$, który wykorzystuje *wygładzanie laplace*.

```
def getClassifireNormal(_features, _targets):
    gnb = GaussianNB()
    gnb.fit(_features, _targets)
    return gnb

def getClassifireSmooth(_features, _targets):
    gnbM = MultinomialNB(alpha=1.0)
    gnbM.fit(_features, _targets)
    return gnbM
```

3 Metody dyskretyzacji

Jednym z celów zadania jest zbadanie wpływu dyskretyzacji danych na jakość klasyfikatora. W programie zaimplementowany zostały trzy rodzaje dyskretyzacji.

1. *Equal width intervals*: podział zakresu wartości atrybutu ciągłego na k przedziałów o jednakowej szerokości.

```
def discretizationEqualWidth(_data, _parts):
    print('Equal width')
    minMaxVals = np.array([])
    transposedData = np.array(_data[:]).transpose()
    resultData = []

    for featureVector in transposedData:
        resultData.append(
            np.digitize(
                featureVector,
                np.linspace(min(featureVector), max(featureVector),
                                                        _parts)
            )
        )

    resultData = np.array(resultData).transpose()
    return resultData
```

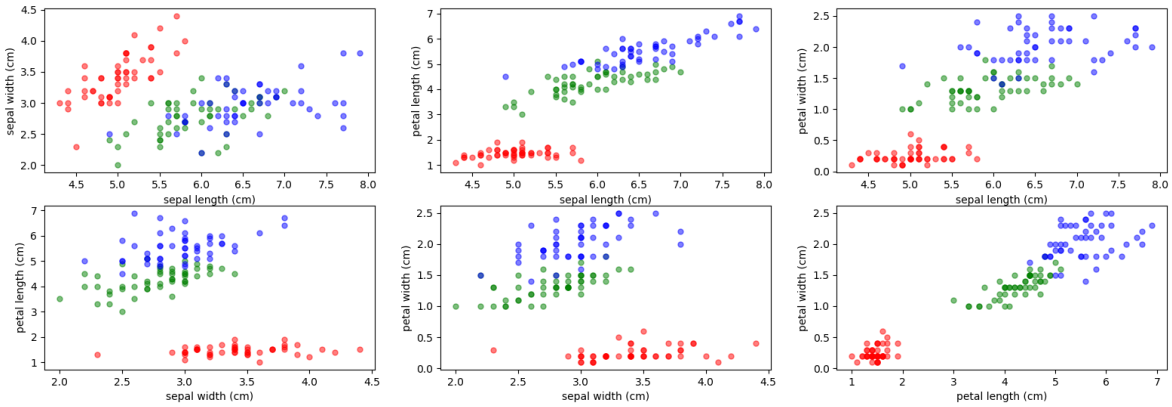
2. *Equal frequency intervals* : podział zakresu wartości atrybutu ciągłego na przedziałów, z których każdemu odpowiada możliwie tyle samo przykładów ze zbioru trenującego.

```
def discretizationEqualFreq(_data):
    print('Equal freq')
    groups = 10
    orangeTable = Orange.data.Table(_data)
    disc = Orange.preprocess.Discretize()
    disc.method = Orange.preprocess.discretize.EqualFreq(n=groups)
    orangeDiscrete = disc(orangeTable)
    mSets = [set() for i in range(0, len(_data[0]))]
    for sample in orangeDiscrete:
        for index in range(0, len(sample)):
            numbers = [float(s) for s in str(sample[index]).split()
                                                    if is_number(s)]
            for numb in numbers:
                mSets[index].add(numb)

    # create bins
    mBins = []
    for mSet in mSets:
        mBins.append(sorted(list(mSet)))


    transposedData = np.array(_data[:]).transpose()
    resultData = []
    for index, featureVector in enumerate(transposedData):
        resultData.append(
            np.digitize(
                featureVector,
                mBins[index]
            )
        )

    resultData = np.array(resultData).transpose()
    return resultData
```

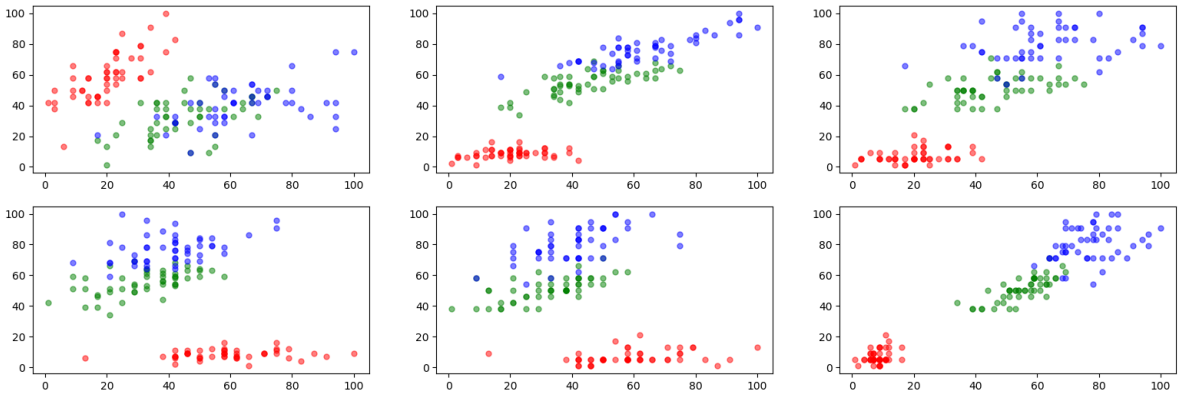
3. *Quantile transform*: konwersja danych do postaci rozkładu normalnego i przypisanie dyskretnych wartości, użyta zostanie funkcja dostępna w pakiecie sklearn.

```
def discretizationQuantileTransform(_data, _quantiles=100):
    print('Quantile')
    nData = list(_data)
    return quantile_transform(nData, n_quantiles=_quantiles,
                                                random_state=0)
```

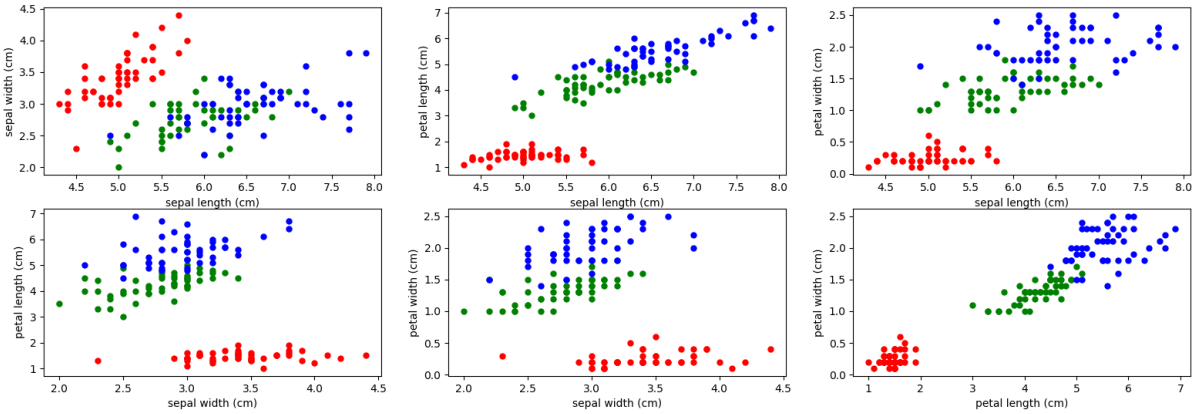
Działanie poszczególnych metod dyskretyzacji dla instancji Iris przedstawiono poniżej.



Rysunek 1: Brak dyskretyzacji



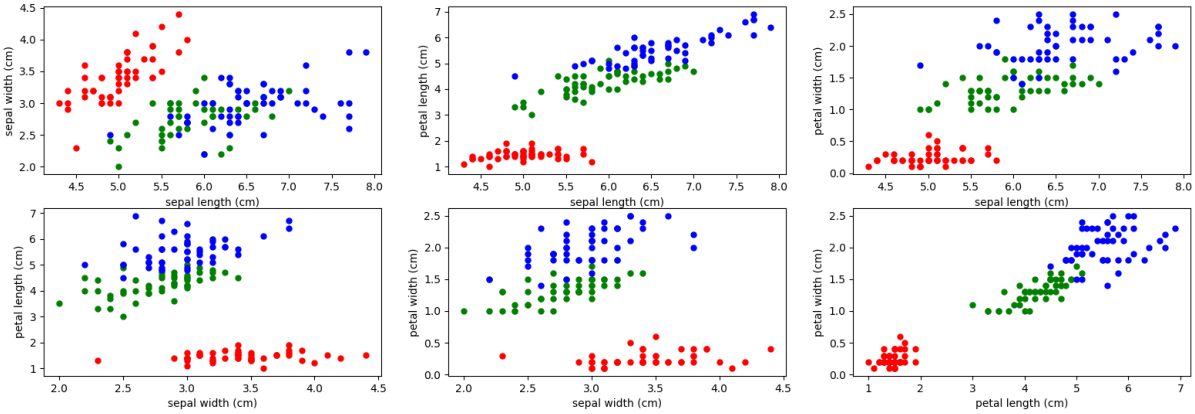
Rysunek 2: Equal width (100 intervals)



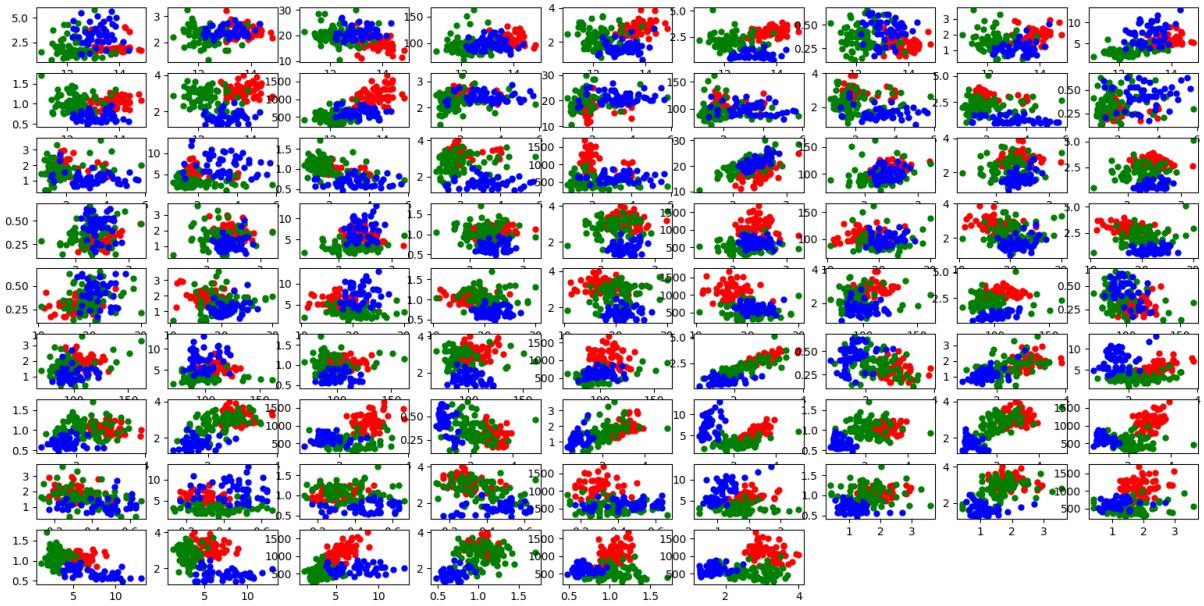
Rysunek 3: Quantile transform

4 Wybór instancji

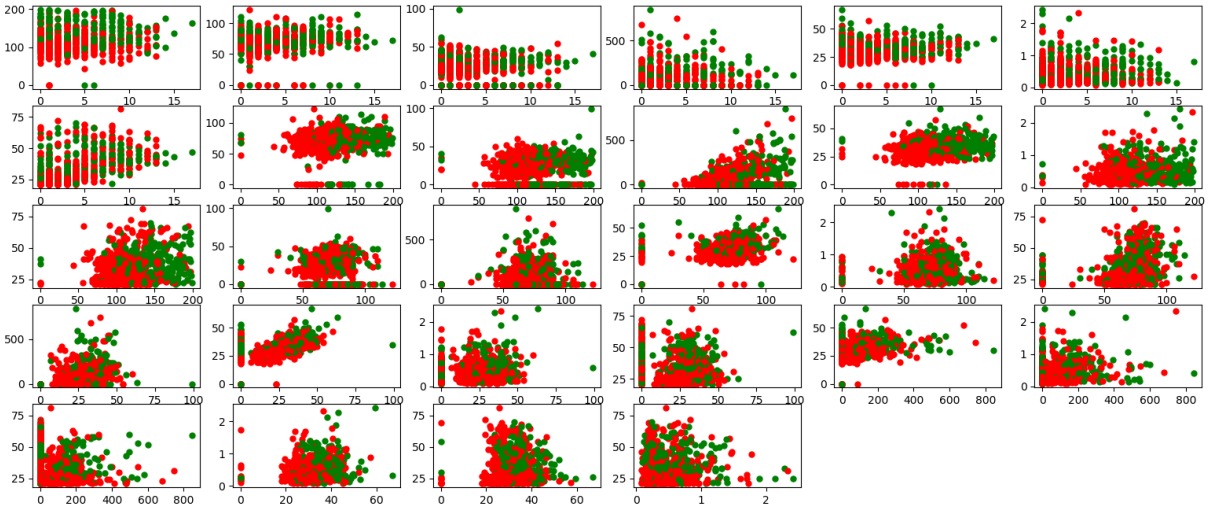
Do testów wykorzystane zostaną gotowe zbiory danych: iris, wine oraz diabetes. Każdy z nich posiada inną ilość cech, które będą uwzględniane w procesie klasyfikacji. Instancje przedstawione zostały na poniższych rysunkach.



Rysunek 4: Instancja Irir



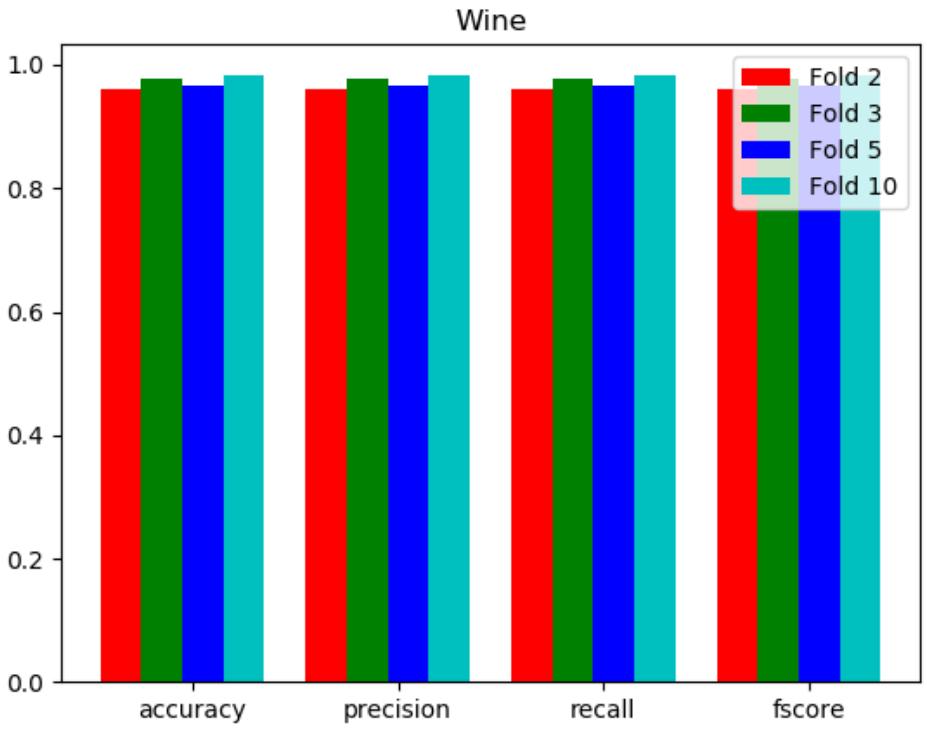
Rysunek 5: Instancja Wine



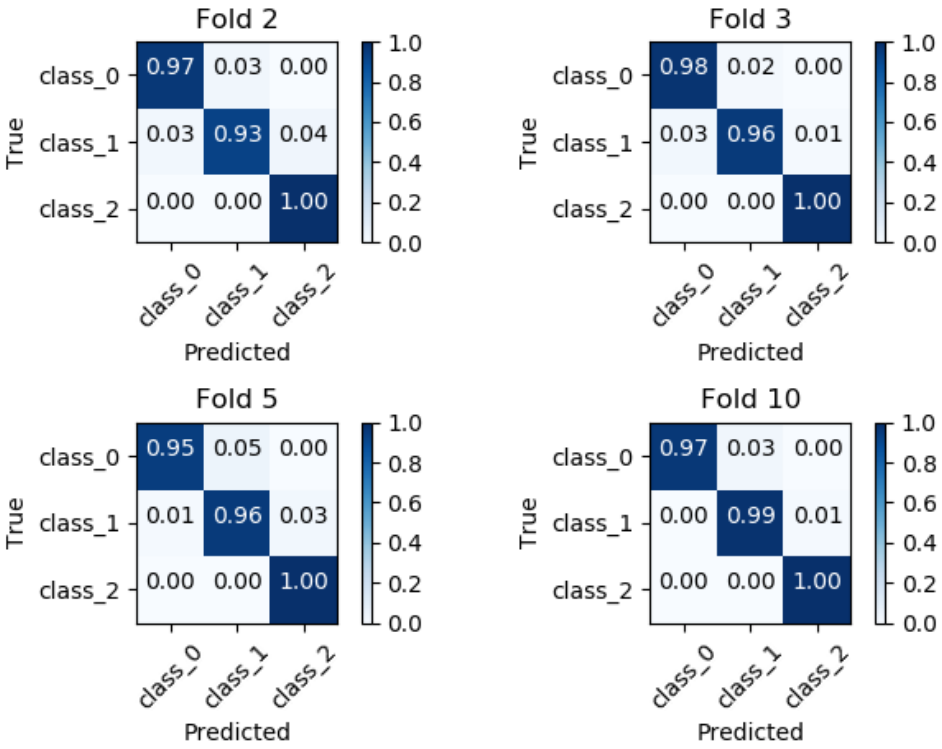
Rysunek 6: Instancja Diabetes

5 Porównanie działania algorytmu przy różnych podziałach danych i sposobie liczenia prawdopodobieństwa

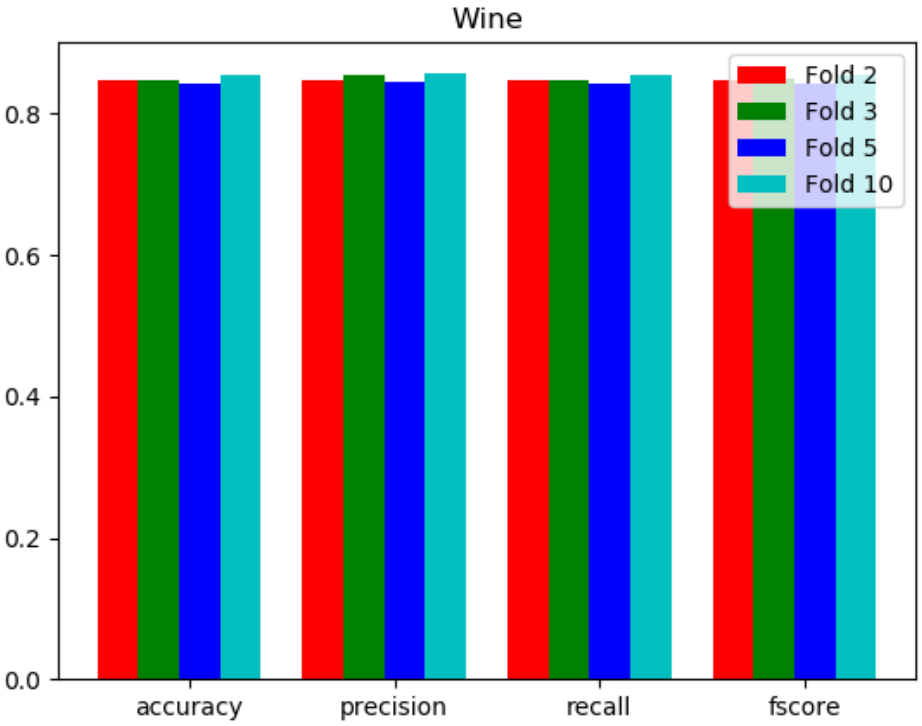
Do oceny klasyfikatora zostanie użyta metoda kroswalidacji z metodą podziału *X-fold*. Polega ona na tym, że zbiór dzielimy na X w miarę możliwości równych części. Każda część zawiera możliwie tyle samo danych z każdej klasy. Jedna część zostanie użyta do oceny klasyfikatora, a pozostałe wejdą w skład zbioru trenującego. Następnie X razy zmianie ulegnie część do klasyfikacji, a cały proces zostanie powtórzony. Przetestowana zostanie metoda uczenia poprzez podział zbioru na 3, 5 i 10 części. Ponadto sprawdzone zostaną dwie metody liczenia prawdopodobieństwa – *Gausian* oraz *Multinomial* z wygładzaniem laplaca.



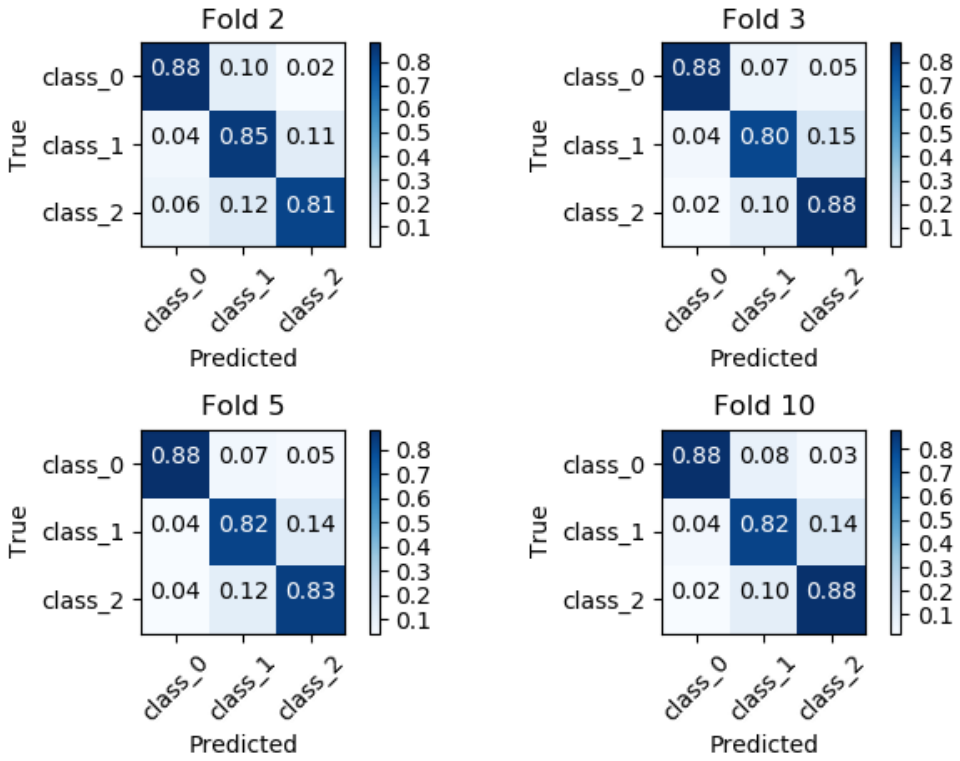
Rysunek 7: Współczynniki dokładności klasyfikatora dla klasyfikatora Gausian



Rysunek 8: Confusion Matrix dla klasyfikatora Gausian



Rysunek 9: Współczynniki dokładności klasyfikatora dla klasyfikatora Multinomial



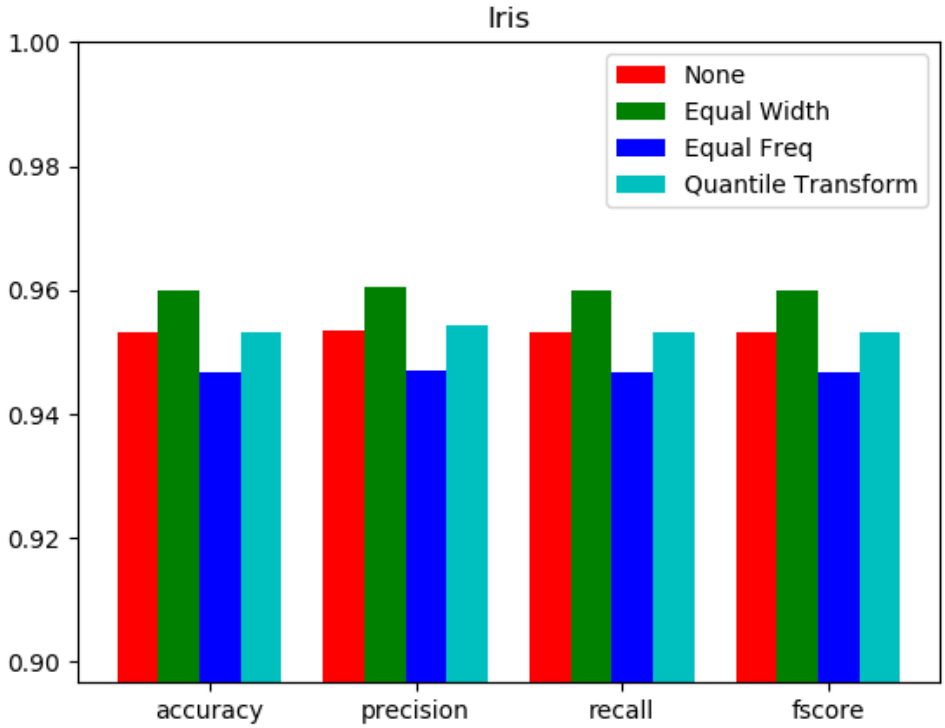
Rysunek 10: Confusion Matrix dla klasyfikatora Multinomial

Można zauważyć, że zmiana ilości podziałów nie ma wpływu na wynik klasyfikacji, co jest spowodowane rozkładem badanych danych. Natomiast zmiana metody liczenia

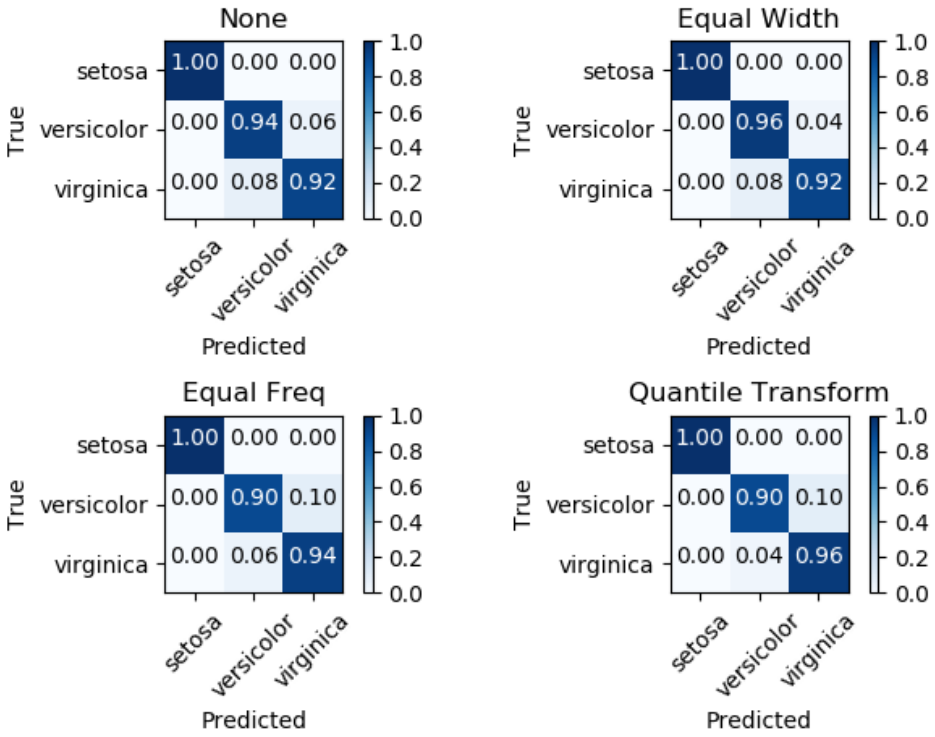
prawdopodobieństwa na sprawiła, że wyniki klasyfikatora uległy znaczącemu pogorszeniu. W późniejszych eksperymentach użyta zostanie metoda 10-fold do walidacji oraz Gausian do liczenia prawdopodobieństwa.

6 Porównanie działania metod dyskretyzacji dla badanych zbiorów

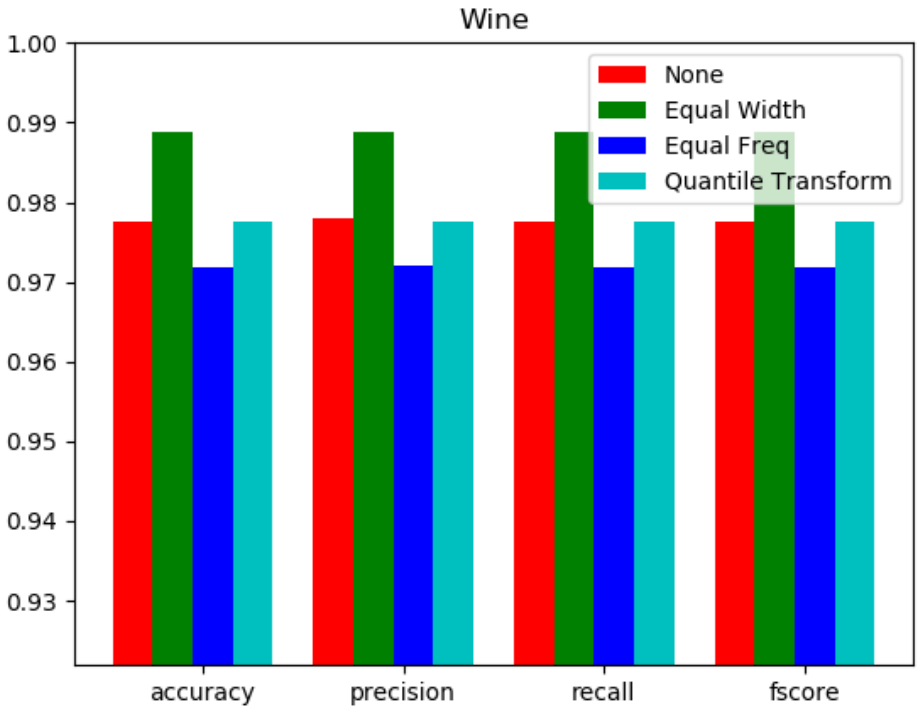
Zaimplementowany klasyfikator zostanie użyty na wybranych wcześniej zbiorach. Dla każdego z nich porównane zostaną wyniki otrzymane przy użyciu różnych metod dyskretyzacji danych. Wyniki przedstawiono na poniższych rysunkach.



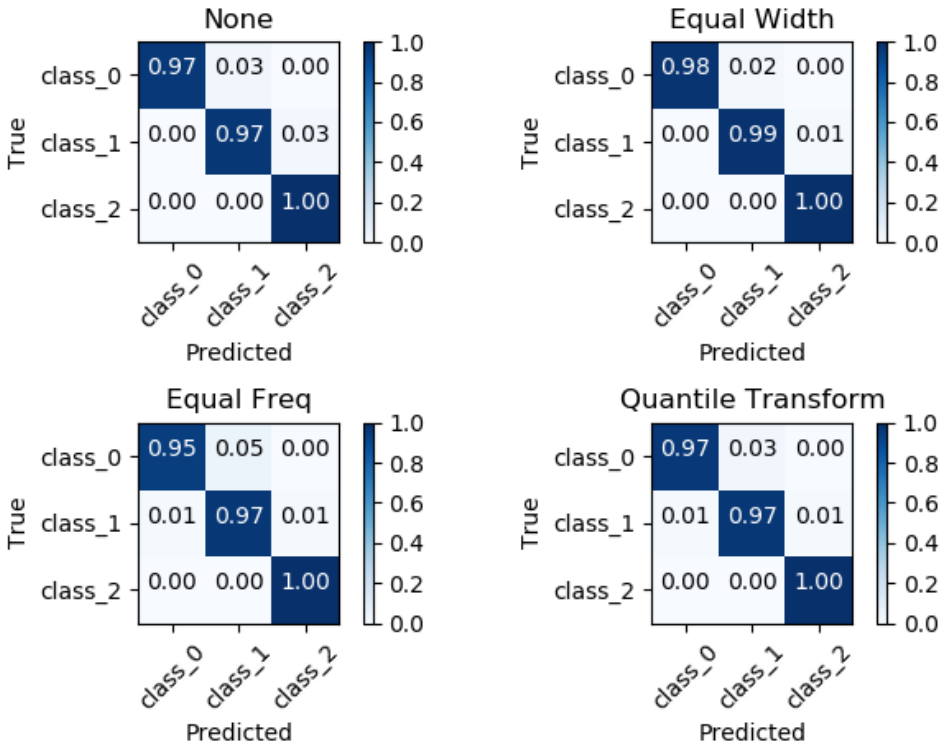
Rysunek 11: Współczynniki dokładności klasyfikatora dla zbioru Iris



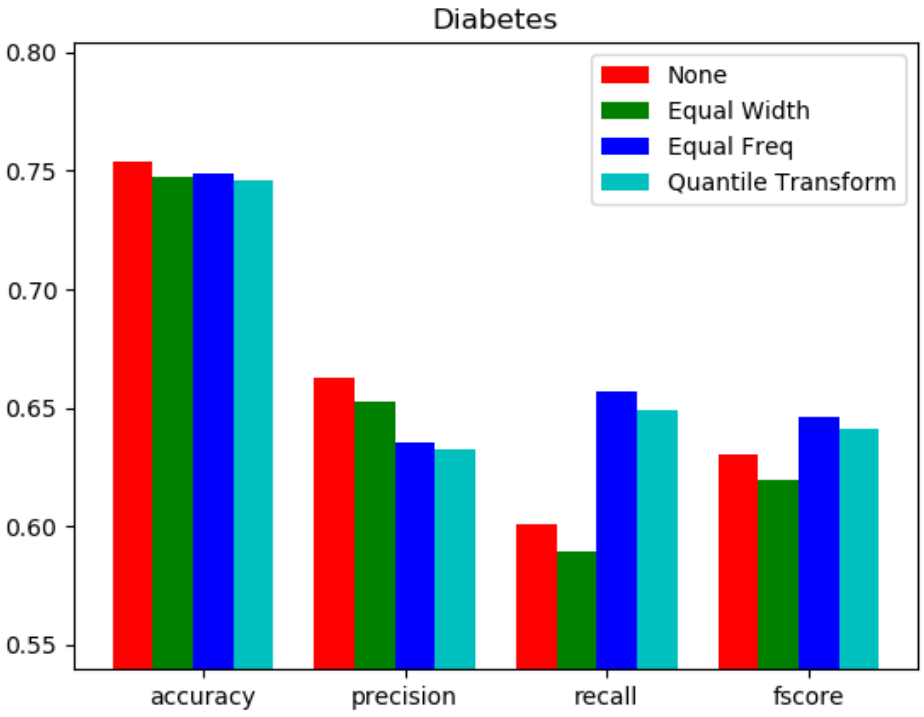
Rysunek 12: Confusion Matrix dla zbioru Iris



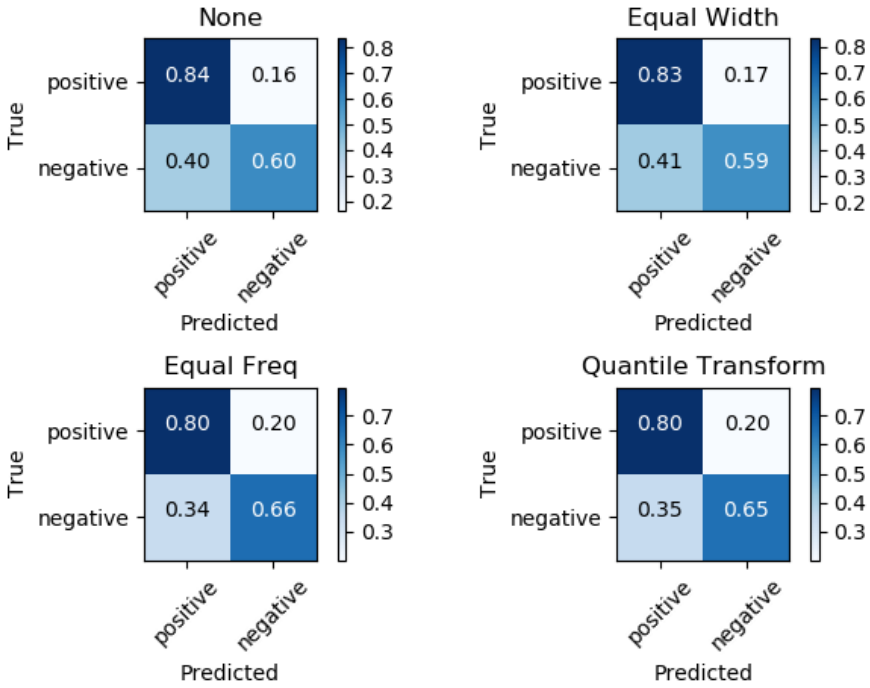
Rysunek 13: Współczynniki dokładności klasyfikatora dla zbioru Wine



Rysunek 14: Confusion Matrix dla zbioru Wine



Rysunek 15: Współczynniki dokładności klasyfikatora dla zbioru Diabetes



Rysunek 16: Confusion Matrix dla zbioru Diabetes

Pomimo zupełnie innych metod dyskretyzacji, jakość klasyfikatorów nie różni się znacząco. W przypadku zbioru *Wine* widać, że najlepsze wyniki osiągnęła metoda równomiernego podziału, ale nie jest to znacząca różnica. W pozostałych przypadkach ciężko jednoznacznie określić najlepszą metodę dyskretyzacji.

7 Wnioski

W ramach zadania zaimplementowany został naiwny klasyfikator bayesa. Przyjęcie, że cechy wybranych zbiorów mają rozkład Gausa dało o wiele lepsze wyniki niż użycie mechanizmu MultinomialBayes. Niezależnie od wykorzystanej metody dyskretyzacji uzyskane klasyfikatory rozpoznawały obiekty ze zbliżoną precyzją. Jest to związane z rozkładem cech danych dla badanych zbiorów.