MPUM - Miniprojekt 2

Michał Hoffmann

1. Zamierzenia projektu

W celu porównania *dyskryminatywnych* i *generatywnych* metod klasyfikacji, na tym samym zbiorze danych wytrenowałem dwa algorytmy: **Regresję logistyczną** oraz **Naiwny klasyfikator Bayesowski**. Końcowo, wyniki zostały porównane z rezultatami zamieszczonymi w artykule o tej samej tematyce.

2. Zastosowane algorytmy

2.1. Regresja logistyczna

Ten klasyfikator wyznacza granicę miedzy dwoma klasami. W celu uzyskania najlepszej predykcji, użyłem prostego spadku gradientowego, nie dodając żadnych hiperparametrów, nie było takiej potrzeby i nie do końca jestem pewny, na ile to odpowiednie .

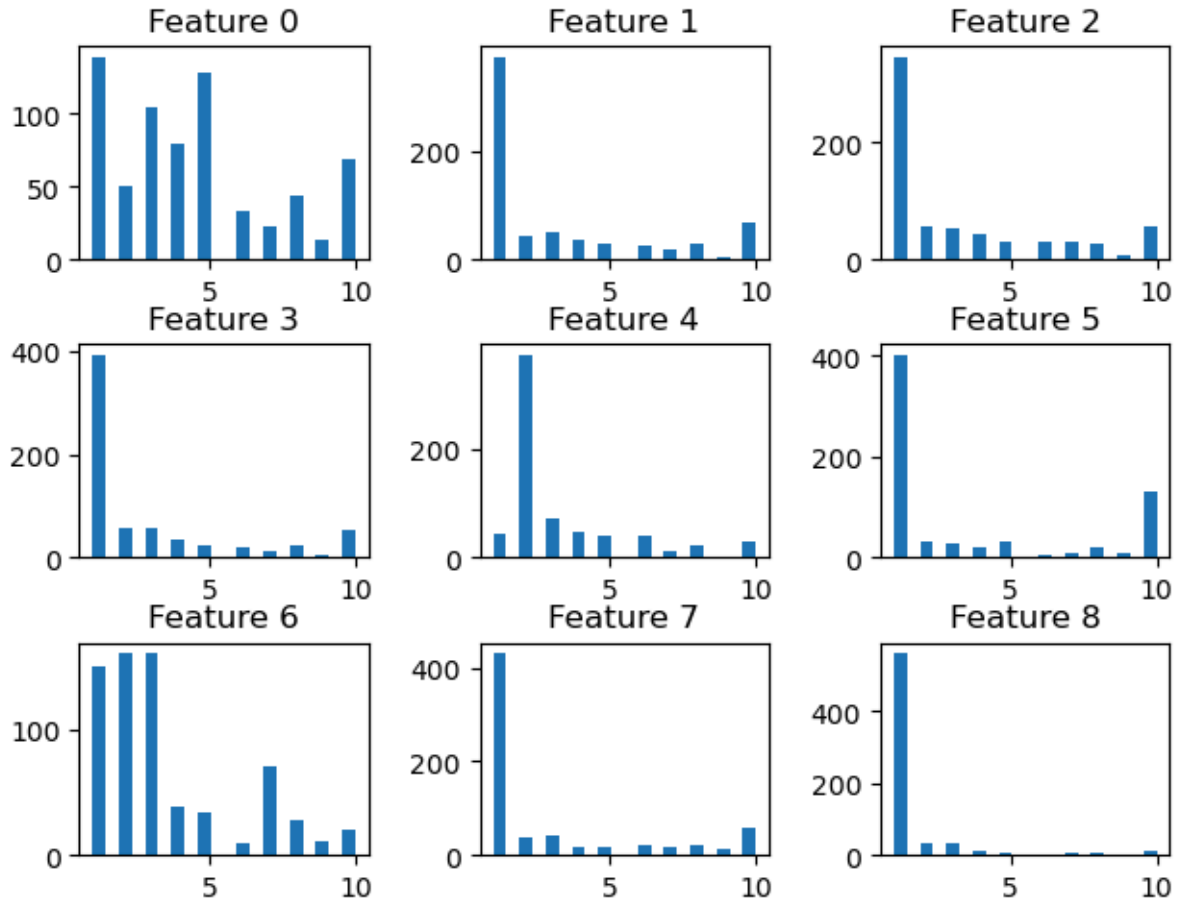
2.2. Naiwny klasyfikator Bayesowski

Metoda klasyfikacji polega na wyznaczeniu prawdopodobieństw a priori dla każdego z parametrów pod warunkiem zmiennej objaśnianej i potem, korzystając ze wzoru Bayesa wyznaczeniu prawdopodobieństwa dla wartości zmiennej objaśnianej. Klasyfikator, z racji mocnego założenia o niezależności parametrów jest naiwny, jednak okazuje się, że w rozważonym przypadku jest bardzo skuteczny.

3. Dane

Dostarczone dane to 9 parametrów $x_i \in \{1, ..., 10\} \mid i \in [9]$. Są to wyniki badań diagnostycznych pacjentów onkologicznych chorujących na nowotwór piersi. Zmienna objaśniana - y, jest binarna i oznacza to, czy nowotwór badanej osoby był złośliwy, czy nie. Początkowo zachodzi $y \in \{2, 4\}$, ale dla wygody przeskalowałem wartości na $\{0, 1\}$, według wzoru: $y' = 0 \Leftrightarrow y = 2$. Ułatwi to implementację algorytmów. Ze względu na charakter danych, najbardziej zależy nam na wykrywaniu jak największej liczby przypadków złośliwych - chcemy zmaksymalizować *recall* (precyzję) modelu.

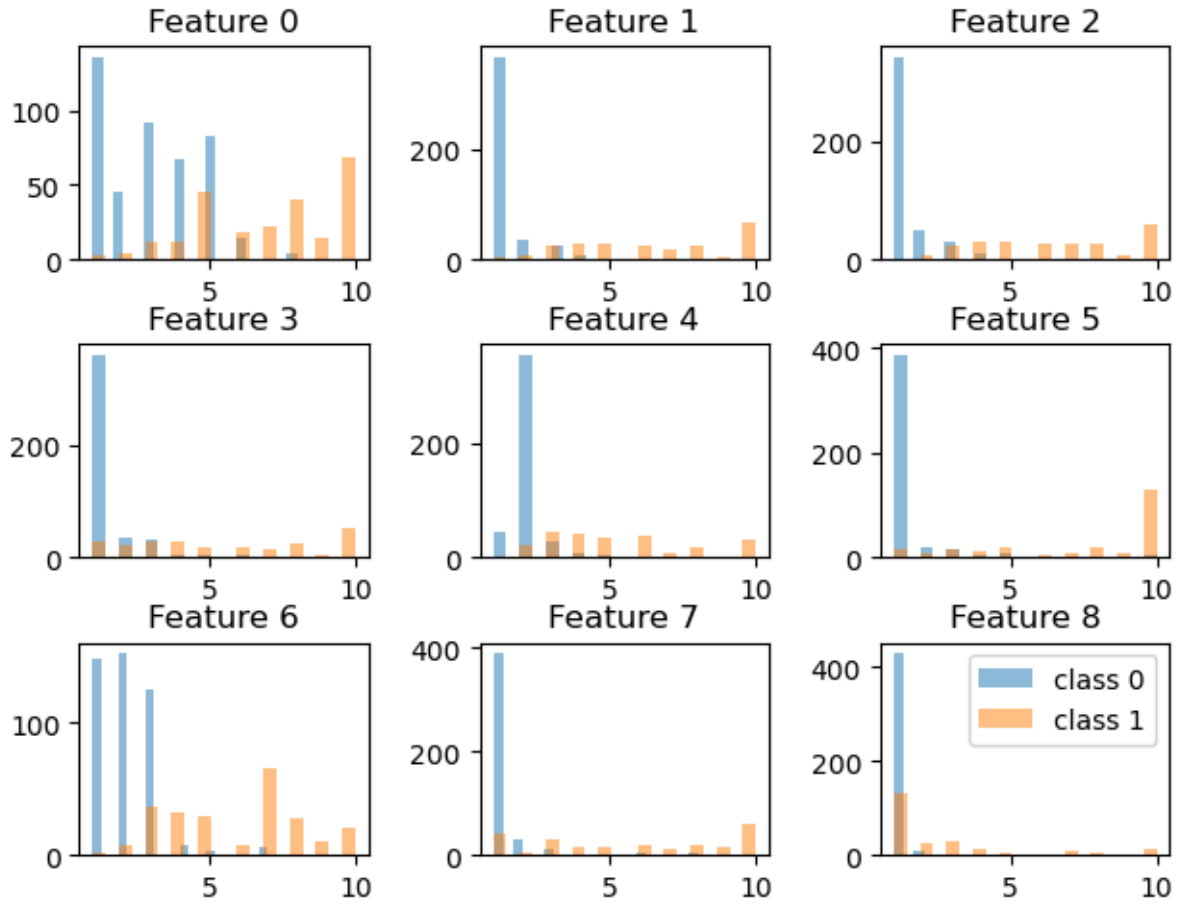
Początkowy rozkład danych względem każdej z cech prezentuje się następująco:



Rysunek 1: Liczba wystąpień każdej z wartości parametrów w danych

Dane podzieliłem na dwa podzbiory X_0, X_1, na podstawie klasy ich zmiennej objaśnianej. Zbiory są niezbalansowane. X_0 stanowi 65% danych, a X_1 - 35%. Istotnym jest to, żeby do analizy używać zbiorów o odpowiedniej liczbie elementów z każdej z klas. Funkcja *split_data* bierze to pod uwagę, dzieląc dane na jeden z dwóch sposobów: funkcja *split_data_evenly* zwraca zbiory treningowy i testowy, w którym elementy każdej klasy stanowią po 50%, a funkcja *split_data_even_proportion* zwraca zbiory o proporcjach równych proporcji w całych danych. Korzystałem głównie z tej drugiej funkcji, gdyż wydaje mi się bardziej realistyczna, zwracają jednak podobne wyniki (porównanie zamieściłem poniżej)

Po podzieleniu na klasy X_0 i X_1 dane rozkładają się trochę inaczej:



Rysunek 2: Liczba wystąpień każdej z wartości parametrów w danych z podziałem na klasy

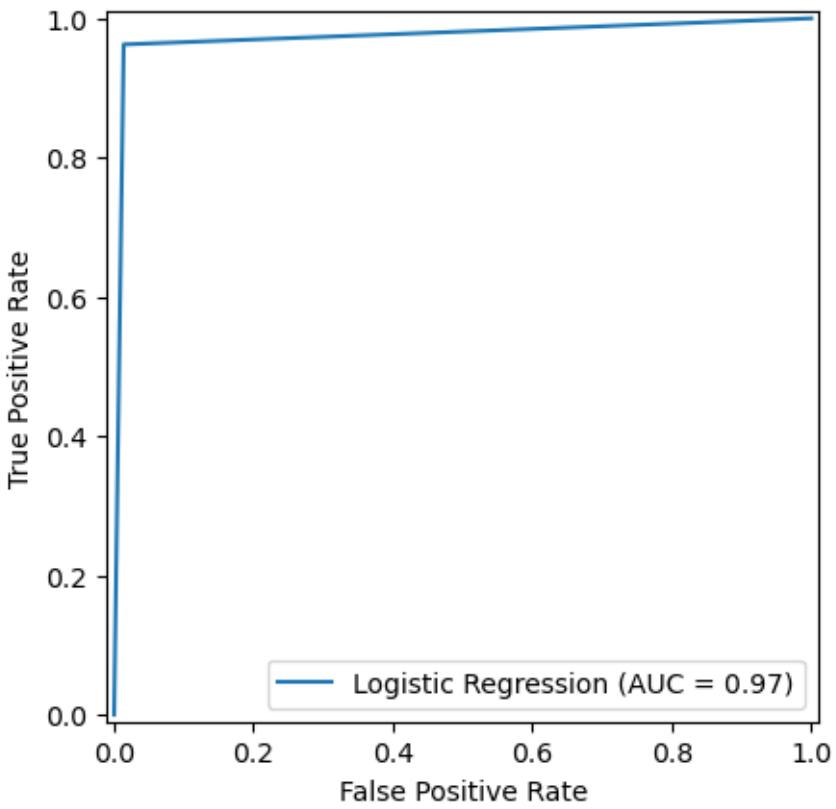
Już tu można zauważyć, że część parametrów będzie bardziej istotna - gołym okiem widać, że klasy cechują różne wartości niektórych parametrów, oznacza to, że nawet w przypadku prostych klasyfikatorów możemy liczyć na dokładne wyniki.

4. Wyniki algorytmów

4.1. Regresja Logistyczna

Na tak małym zbiorze cech jak w dostarczonych danych, algorytm gradientu wykonywał się niemalże natychmiastowo. Postanowiłem nie dodawać nowych cech. Nie jestem pewny, czy w ogóle jest to dozwolone, a i wyniki, w przypadku dodania pozornie wartościowych cech, wcale nie były lepsze. Tak samo zrezygnowałem z regularyzacji, bo rozważając wykresy uczenia na ułamkach zbioru testowego, algorytm nie wyglądał przetrenowany. Koniec końców, jedyna modyfikacja którą zastosowałem, to przesunięcie granicy decyzyjnej, żeby wykrywać więcej przypadków złośliwych. Granica została obniżona do 0.35, znacznie polepszyło to te parametry, które chciałem zmaksymalizować.

4.1.1. krzywa ROC

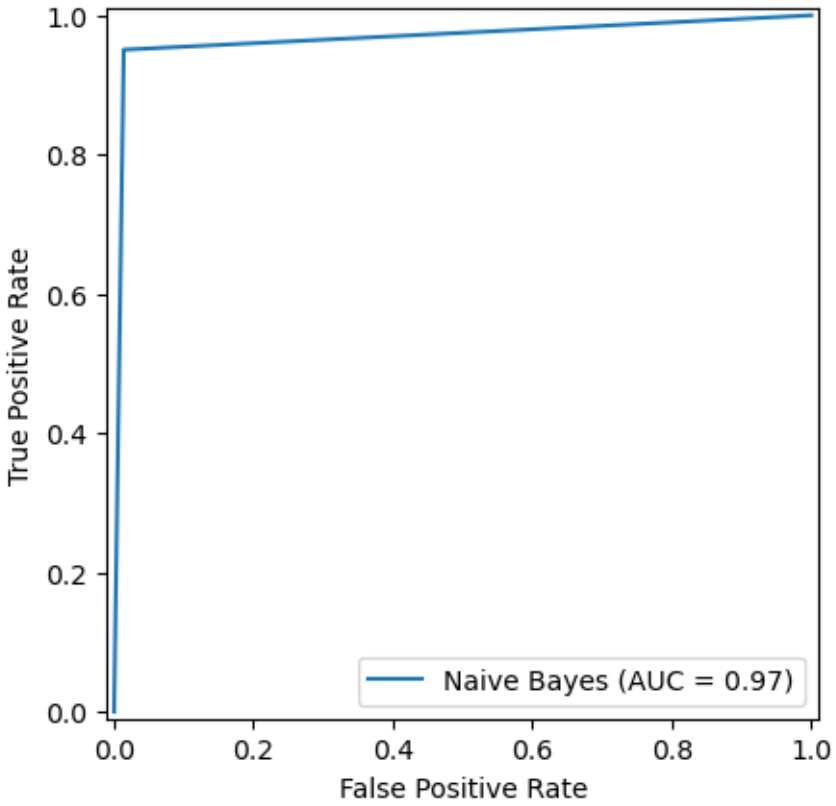


Rysunek 3: krzywa ROC dla regresji logistycznej (gotowa funkcja z biblioteki sklearn.metrics)

4.2. Naiwny Bayes

Poza zalecanym wygładzeniem Laplace'a, jedyną zmianą, która delikatnie poprawiła jakość wyników, było przesunięcie granicy decyzyjnej na 0.40. Eksperymentalnie sprawdziłem, że polepsza to jakość tych wyników, na których zależy nam najbardziej (accuracy i recall), małym kosztem precyzji.

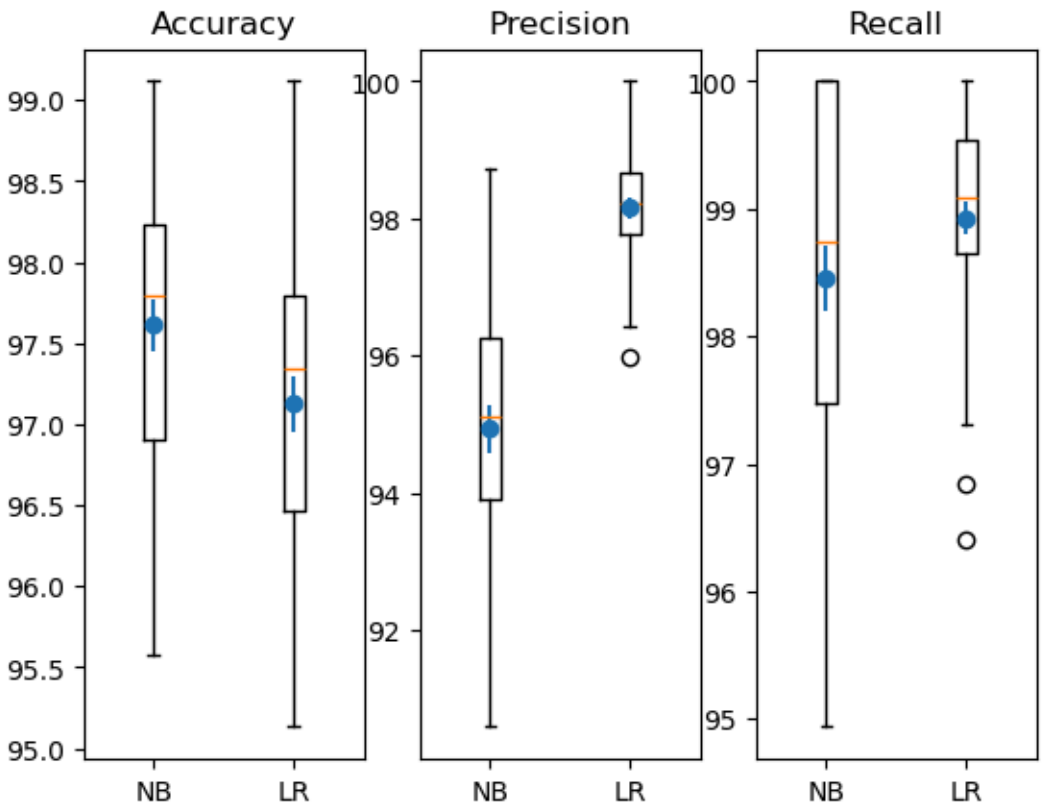
4.2.1. krzywa ROC



Rysunek 4: krzywa ROC dla regresji logistycznej (gotowa funkcja z biblioteki sklearn.metrics)

4.3. Końcowe porównanie wyników

Dla stu przebiegów, średnia prezentuje się następująco:



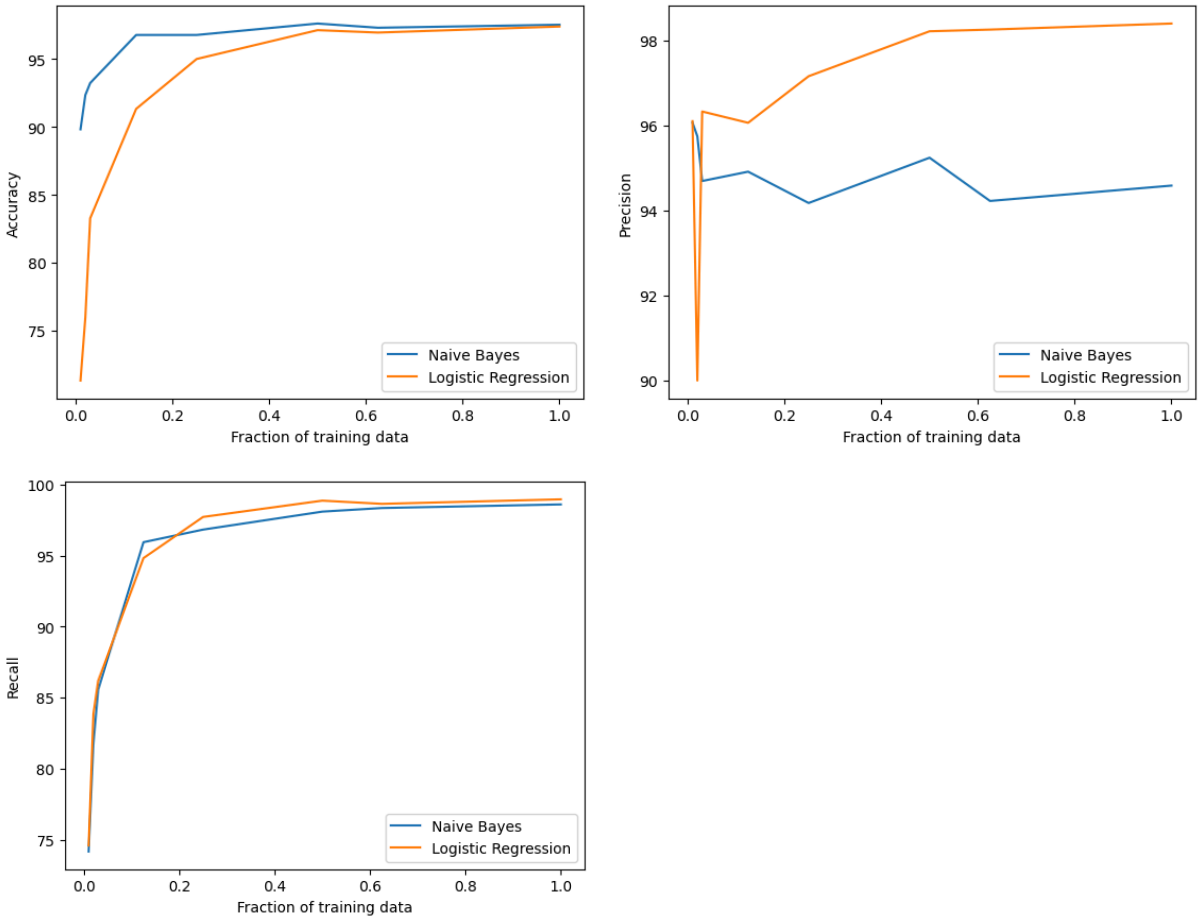
Rysunek 5: Uśrednione wyniki klasyfikacji. NB- Naive Bayes, LR- Logistic Regression

W przypadku rozpoznawania nowotworu złośliwego, wykresy sugerują, żeby stosować regresję logistyczną, choć na pierwszy rzut oka to Bayes jest lepszym klasyfikatorem.

5. Proces uczenia na fragmentach zbioru treningowego

Według dołączonego artykułu, klasyfikator Bayesowski powinien działać dobrze już dla ułamka danych $p \in O(\log n)$, niestety, zgodnie z eksperymentami, mój klasyfikator potrzebuje trochę więcej danych, gdyż dla dostarczonego zbioru, $\log n$ danych, to około 1.5% zbioru testowego, kiedy Bayes osiąga przyzwoite wyniki dopiero po rozważeniu około 10% zbioru.

Poniższe wykresy to uśrednienie 10 przebiegów na losowych próbkach zbioru treningowego



Rysunek 6: Wyniki dla ułamków zbioru treningowego.

Jak widać, krzywa uczenia zgadza się z wynikami z artykułu - różnica między regresją a Bayesem dla małych ułamków jest większa niż dla pełnych danych, jednakże w moim przypadku, regresja w ogólnej co najwyżej dorównała do klasyfikatora Bayesowskiego nawet dla pełnych danych. Jedynym benchmarkiem, gdzie sprawowała się dużo lepiej była precyzja, ale w naszym przypadku najważniejszy jest *recall*, który był porównywalny dla obu klasyfikatorów na krzywej, ale jak pokazałem wyżej, średnio Regresja wypadała odrobinę lepiej.

6. Porównanie z gotowymi rozwiązaniami

Jako benchmarka użyłem gotowych klas: *GaussianNB* z *sklearn.naive_bayes* i *LogisticRegression* z *sklearn.linear_model*:

```
Naive Bayes
Accuracy:  0.9611504424778761
Precision:  0.9805063291139241
Recall:  0.9155660451032313
Logistic Regression
Accuracy:  0.9665929203539824
Precision:  0.9496202531645568
Recall:  0.9551788818457951
```

Rysunek 7: Wyniki benchmarków (średnia 100 iteracji)

Zaskakująco, moje wyniki były nawet odrobinę lepsze niż te, z gotowych funkcji. Moje obliczenia zajmowały natomiast kilkukrotnie więcej czasu. Mimo wszystko, uznaję to za duży sukces. W szczególności, recall w moim przypadku był dużo lepszy, a jak już wspominałem, starałem się skupiać na jak najlepszym wyniku właśnie tej statystyki.