Uczenie Maszynowe

Końcowe sprawozdanie z Projektu

Temat projektu: Techniki oceny klasyfikacji dla zestawów danych dotyczących raka piersi

Mateusz Krakowski

Bartosz Latosek

10 maja 2023

Spis treści

1 Opis Projektu

1.1 Treść zadania

Zaimplementuj techniki oceny klasyfikacji dla zestawów danych dotyczących raka piersi, które dostępne są w: http://archive.ics.uci.edu/ml/datasets

1.2 Użyte dane

Wykorzystaliśmy dane z datasetu Breast Cancer Data Set [LINK].

1.3 Analiza danych

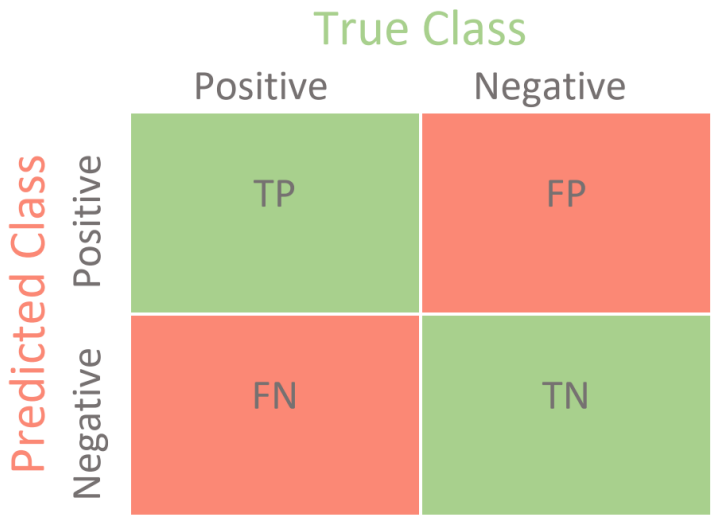
Pełna analiza danych znajduje się w pliku data_analisis.ipynb, tutaj zamieszczamy skrót naszych odkryć. Wniski z analizy danych:

- posiadamy dane 286 osób, jest to mało aby nauczyć dobry klasyfikator, ale nie będziemy się w tym projekcie na samym uczeniu dobrych klasyfikatorów, a na wizualizacji miar jakości modeli.
- Klasą większościową są osoby u których nie wystąpiły zdarzenia rekurencyjne (no-recursive-events), stanowią 70% całego datasetu. Reszta to osoby u których wystąpiły zdarzenia rekurencyjne.
- Udało nam się zauważyć silną korelację między wiekiem a posiadaniem menopałzy ge40 oraz niski wiek jest skorelowany z posiadaniem menopauzy premeno.
- Najbardziej skorelowane z atrybutem klasy są: deg-malig, node-caps, inv-nodes. Jest to jednak korelacja na poziomie 0.3, dodatkowo trzeba pamiętać że korelacja nie oznacza przyczynowości.

1.4 Miary jakości modelu

1.4.1 Macierz pomyłek

Macierz zawierająca 4 wartości mówiące o tym jak model poradził sobie z klasyfikacją danych



Rysunek 1: Macierz Pomyłek

Na podstawie macierzy pomyłek obliczane poniższe miary jakości.

1.4.2 Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

Accuracy niesie informację o tym, jaki procent próbek testowych został poprawnie sklasyfikowany przez model.

1.4.3 Recall(Sensitivity)

$$Recall = Sensitivity = \frac{TP}{TP + FN}$$

Recall mówi nam, jak model radzi sobie z klasyfikowaniem przypadków pozytywnych danej klasy.

1.4.4 Specificity

$$Specificity = \frac{TN}{TN + FP}$$

Specificity mówi nam, jak model radzi sobie z klasyfikowaniem przypadków negatywnych danej klasy.

1.4.5 Precision

$$Precision = \frac{TP}{TP + FP}$$

Precision mówi nam, w jakich proporcjach model klasyfikuje próbki jako pozytywne w zależności od faktycznej klasy próbki.

1.4.6 Miara F1

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

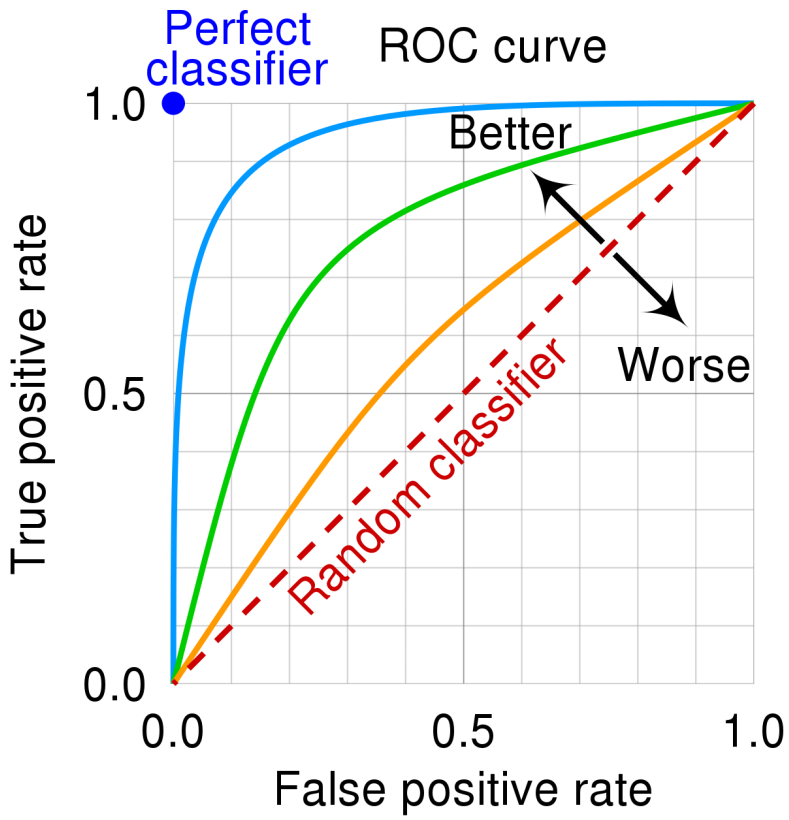
F1 score mówi nam, jak dobrze model radzi sobie z klasyfikacją przypadków TP, przy tym minimalizując liczbę przypadków FP i FN.

1.4.7 Support

$$Support = TP + FN$$

Support mówi nam ile w danych ewaluacyjnych jest osób posiadających rekursywne zdarzenia.

1.4.8 Krzywe ROC i AUC



Rysunek 2: Krzywa ROC

Krzywa rok pokazuje nam zależność TPR(true positive rate) od FPR(false positive rate).

$$TPR = Recall = Sensitivity = \frac{TP}{TP + FN}$$

$$FPR = 1 - Specificity = \frac{FP}{FP + TN}$$

Rysuje się ją poprzez sprawdzenie, jak algorytm klasyfikuje przypadki dla wybranych poziomów cutoff. Cutoff to liczba z zakresu [0, 1] która definiuje które predykcje modeli są klasyfikowane jako klasa pozytywna lub negatywna. AUC to pole pod wykresem narysowanym w powyższy sposób.

1.5 Założenia odnośnie metryk oceny

W naszym zadaniu samo accuracy nie będzie dobrym wyznacznikiem jakości modelu. Ważne będzie, aby przy ocenie wziąć pod uwagę Sensitivity oraz Specificty. To, aby ustalić która z tych dwóch metryk jest ważniejsza, musi odpowiedzieć pytanie czy jesteśmy bardziej skłonni dopuścić do klasyfikacji przypadków fałszywie negatywnych, czy fałszywie pozytywnych. Osobiście zdaje mi się, że większe straty ponosimy w przypadku klasyfikacji FN, przypadki FP zawsze można wykluczyć w dogłębnym badaniu, na przykład sięgając po opinię specjalistów lub innego algorytmu. Dlatego zdaje mi się, że specificity będzie w tym problemie ważniejszą metryką. Dodatkowo dobrymi miarami do porównania algorytmów będzie miara F1 oraz krzywe ROC i AUC.

1.6 Modele badane miarami jakości

1.6.1 Model losowy

Model losowy zwracający wartość losową z zakresu [0, 1], oznaczającą prawdopodobieństwo sklasyfikowania osobnika jako przynależnego do klasy 1 (wystąpiły zdarzenia rekurencyjne).

1.6.2 Naiwny Bayes

Naiwny klasyfikator Bayesowski to metoda klasyfikacji, która opiera się na teorii Bayesa. Pozwala ona na przypisanie nowych obiektów do jednej z kilku klas na podstawie cech, które posiadają.

Działanie naiwnego klasyfikatora Bayesowskiego polega na wyznaczeniu prawdopodobieństwa przynależności nowego obiektu do każdej z klas na podstawie jego cech. **Klasyfikator zakłada, że cechy obiektów są niezależne od siebie**, co może być uproszczeniem, ale w wielu przypadkach działa wystarczająco dobrze.

Aby wyznaczyć prawdopodobieństwo przynależności nowego obiektu do danej klasy, naiwny klasyfikator Bayesowski korzysta z wcześniej zebranych danych treningowych, w których już zostały przyporządkowane klasy dla innych obiektów o znanych cechach. Dzięki temu można wyznaczyć, jakie cechy są typowe dla danej klasy i jakie prawdopodobieństwo przynależności do niej mają obiekty posiadające te cechy. W celu uproszczenia stworzenia modelu wykorzystaliśmy GaussianNB z modułu *sklearn.naive_bayes*.

1.6.3 Random Forest

Główną ideą Random Forest jest połączenie wielu drzew decyzyjnych, które są trenowane na różnych podzbiorach danych treningowych, wykorzystując losowe próbkowanie ze zwracaniem. Każde drzewo w lesie ma swoją niezależną losowość, co oznacza, że losuje tylko pewną liczbę cech spośród wszystkich dostępnych. W ten sposób tworzone są różnorodne drzewa, które uzupełniają się nawzajem.

W drzewie decyzyjnym kolejne warunki są reprezentowane jako węzły, a decyzje jako gałęzie, które łączą węzły. Drzewo zaczyna się od węzła nadrzędnego, nazywanego korzeniem, który reprezentuje główne pytanie.

Z każdego węzła wychodzą gałęzie prowadzące do kolejnych węzłów, które reprezentują różne możliwe decyzje lub wyniki. Przechodząc w dół po drzewie, na każdym etapie podejmujemy decyzje na podstawie cech lub zmiennych, aż osiągniemy końcowy węzeł, który reprezentuje ostateczną decyzję lub w naszym przypadku - klasę. W celu uproszczenia stworzenia modelu wykorzystaliśmy RandomForestClassifier z modułu *sklearn.ensemble.RandomForestClassifier*.

1.6.4 XGBoost

Gradient boosting to technika uczenia maszynowego, która polega na budowaniu sekwencji słabych modeli predykcyjnych i łączeniu ich w silny model. Słabe modele są trenowane iteracyjnie w celu minimalizacji funkcji kosztu, a każdy kolejny model dostosowuje się do reszt, jakie pozostawiają poprzednie modele.

XGBoost implementuje tę technikę w sposób zoptymalizowany pod kątem wydajności i skuteczności. W przeciwieństwie do innych metod gradient boosting, XGBoost wykorzystuje regresję logistyczną jako funkcję kosztu, co zapewnia stabilność procesu uczenia. W celu uproszczenia stworzenia modelu wykorzystaliśmy XGBoost z modułu *xgboost.xgb*.

1.6.5 Sieć neuronowa

W celu uproszczenia zdecydowaliśmy się na implementację własnej sieci neuronowej. Posiada ona dwie warstwy ukryte i jedną warstwę wyjściową. W celu optymalizacji i zapobieganiu przeuczeniu - pomiędzy warstwami ukrytymi wykorzystaliśmy metodykę dropoutu oraz batchnorm.

1.6.6 Support Vector Machines

SVM to algorytm który dąży do odnalezienia hiperpłaszczyzny która najlepiej odseparowuje dane należące do obydwu klas. Hiperpłaszczyzna ta ma maksymalizować margines, czyli minimalną odległość między hiperpłaszczyzną a najbliższymi punktami z obu klas.

2 Struktura projektu

2.1 Dane

Wszystkie modele klasyfikujące raka piersi, są trenowane a następnie sprawdzane przy użyciu danych, które dostępne są w:

http://archive.ics.uci.edu/ml/datasets.

Do reprezentacji danych w projekcie używamy abstrakcji w postaci klasy Data. W niej jako atrybuty przechowywane są surowe dane jak i obrobione dane, wymagane przez niektóre z modeli. W wyniku obróbki danych atrybuty nominalne zakodowane są za pomocą one-hot encoding. Dodatkowo uprościliśmy dane reprezentowane przez zakres do pojedynczej wartości będącej środkiem przedziału. Następnie wszystkie tak obrobione dane zostały znormalizowane do zakresu [0, 1]. Do reprezentacji danych w projekcie używamy abstrakcji w postaci klasy Data. W niej jako atrybuty przechowywane są surowe dane jak i obrobione dane, wymagane przez niektóre z modeli. W wyniku obróbki danych atrybuty nominalne zakodowane są za pomocą one-hot encoding. Dodatkowo uprościliśmy dane reprezentowane przez zakres do pojedynczej wartości będącej środkiem przedziału. Następnie wszystkie tak obrobione dane zostały znormalizowane do zakresu [0, 1].

3 Opis Funkcjonalny

Poniżej przedstawiona zostanie struktura projektu.

3.1 Dane

Do operacji na danych takich jak preprocessing, czy podział na zbiór testowy i walidacyjny służy klasa Data pakietu data.

Analiza danych pod kątem atrybutów została przeprowadzona na podstawie pliku:

data_analysys.ipynb

3.2 Modele

Wszystkie modele w projekcie dziedziczą po abstrakcyjnej klasie BinaryClassificationModel, wprowadzającej wygodny w użyciu interfejs do późniejszej analizy i oceny jakości modeli. Wszystkie modele wewnętrznie wyznaczają miary oceny jakości a następnie przechowują je w atrybutach, do których w prosty sposób można się dostać. Posiadają one również metody służace do rysowania confusion matrix oraz ROC curve.

W projekcie zaimplementowane zostały następujące modele, znajdujące się w plikach odpowiadającym ich nazwom:

1. Model losowy
2. Naiwny Bayes

3. Sieć neuronowa
4. Random Forest
5. XGBoost
6. Support Vector Machine

3.3 Main

W notatniku *main.ipynb* znajduje się najważniejsza część projektu, łącząca ze sobą wszystkie modele i przedstawiająca wyniki analizy miar jakości testowanych modeli.

3.4 Testy jednostkowe

Stworzyliśmy testy jednostkowe w celu przetestowania działania naszych klas oraz metod. Dodatkowo w trakcie pisania testów okazało się, że nie obsłużyliśmy błędu polegającego na tym że dla niektórych danych, niektóre funkcje obliczające metryki zwracały błąd DevisionByZeroError, obsłużyliśmy błędy, gdy miałoby dojść do dzielenia przez zero to wtedy metryka nie jest obliczana, pozostaje Nullem. Testy jednostkowe dodatkowo dobrze pokazują jak należy korzystać z klas które przygotowaliśmy. Testy znajdują się w plikach test_BinaryClassificationModel.py i data/test_Data.py.

4 Wnioski końcowe

4.1 Wstępne założenia

W naszym zadaniu samo accuracy nie będzie dobrym wyznacznikiem jakości modelu. Ważne będzie, aby przy ocenie wziąć pod uwagę Sensitivity oraz Specificty. To, aby ustalić która z tych dwóch metryk jest ważniejsza, musi odpowiedzieć pytanie czy jesteśmy bardziej skłonni dopuścić do klasyfikacji przypadków fałszywie negatywnych, czy fałszywie pozytywnych. Osobiście zdaje mi się, że większe straty ponosimy w przypadku klasyfikacji FN, przypadki FP zawsze można wykluczyć w dogłębnym badaniu, na przykład sięgając po opinię specjalistów lub innego algorytmu. Dlatego zdaje mi się, że specificity będzie w tym problemie ważniejszą metryką. Dodatkowo dobrymi miarami do porównania algorytmów będzie miara F1 oraz krzywe ROC i AUC.

4.2 Wpływ danych na wyniki

Niestety, zbiór danych na którym operujemy jest mały, zaledwie 286 obserwacji. Dlatego można się spodziewać, ze wariancja wyników będzie duża i nie będzie można jednoznacznie stwierdzić, który algorytm jest najlepszy. Zważając na to, że nasz projekt miał skupić się na inplementacji algorytmów, a nie porówaniu wyników, w raporcie zawarte zostaną tylko ciekawe obsercaje, a nie pełna analiza statystyczna.

4.3 Obserwacje

- Dobrze w tym problemie poradził sobie Naiwny Bayes, często osiągał największe specificity i F1 score. Przypuszczamy, że jest to spowodowane tym, że dane nie są ze sobą skorelowane, a algorytm NB zakłada niezależność zmiennych.
- Najlepsze specificity najczęściej osiągał algorytm SVM, jednak często osiągał też najgorsze wyniki pod względem sensitivity i gorsze F1 od modelu losowego.
- Sieci neuronowe nie wypadły w naszym projekcie najlepiej, przypuszczamy, że jest to spowodowane małą ilością danych.

4.4 Wniosek końcowy

Jeśli miałbym wybrać najlepszy model do rozwiązania naszego problemu, nie wybierał bym wcale. Niestety posiadamy za mało danych aby wytrenować dobry, skomplikowany model. Dla tak małego zmioru danych, dobrymi roziwązaniami okazały się takie modele jak losowy las drzew decyzyjnych czy naiwny bayes.

5 Użyte narzędzia i biblioteki

Projekt wykonaliśmy w języku programistycznym Python. Wykorzystaliśmy biblioteki:

- matplotlib==3.7.1 - do wykresów
- numpy==1.22.0 - do obliczeń
- pandas==1.3.5 - do działań na wektorze danych
- scikit_learn==1.1.2 - do niektórych modeli
- seaborn==0.11.2 - do wizualizacji danych
- torch==1.12.1 - do obliczeń na tensorach i sieci neuronowej
- xgboost==1.5.2 - do gotowej implementacji xgboost
- pytest==7.3.1 - do testów