Identyfikacja choroby Parkinsona na podstawie mowy.

KWD

Jakub Brzeżański i Tomasz Zapadliński

21.12.2022

Streszczenie

W niniejszym projekcie przeprowadziliśmy badanie nad wykorzystaniem uczenia maszynowego do wykrywania choroby Parkinsona. W celu oceny skuteczności modelu regresji logistycznej zastosowano dane z badań klinicznych, zawierające parametry kilkuset nagrań głosowych. Zastosowaliśmy różne narzędzia manipulujące ilością cech oraz dobierające odpowiednie metody regularyzacji w celu wybrania jak najbardziej optymalnego modelu oraz poznania problemu i danych ,którymmi dysponujemy. Rezultaty wskazują, że algorytmy uczenia maszynowego mogą być skuteczne w wykrywaniu choroby Parkinsona z wysoką dokładnością, co sugeruje, że mogą one być użyteczne w diagnostyce tej choroby.

Spis treści

Rozdział 1

Wprowadzenie

Choroba Parkinsona jest samoistną, powoli postępującą, zwyrodnieniową chorobą ośrodkowego układu nerwowego, należącą do chorób układu pozapiramidowego. Nazwa pochodzi od nazwiska londyńskiego lekarza Jamesa Parkinsona, który w 1817 roku rozpoznał i opisał objawy tego schorzenia. Wczesne rozpoznanie choroby Parkinsona może pomóc w leczeniu, jednak rozpoznanie może być trudne, ponieważ objawy mogą być subtelne i różne wsród wszystkich pacjentów

1.1 Opis problemu

Uczenie maszynowe i regresja logistyczna mogą być używane do rozpoznawania choroby Parkinsona na podstawie próbek głosu pacjentów. Regresja logistyczna jest metodą statystyczną, która służy do przewidywania prawdopodobieństwa danej etykiety na podstawie danych wejściowych. Nadaje się więc do rozpoznania choroby.

Do odpowiedniego wyuczenia takiego modelu regresji logistycznej, należy najpierw zgromadzić dużą liczbę próbek głosu od osób z chorobą Parkinsona i osób zdrowych. Następnie należy zastosować odpowiednie narzędzie do analizy głosu, takie jak analiza częstotliwości lub analiza mocy głosu, w celu wyodrębnienia cech głosu, które mogą być charakterystyczne dla choroby Parkinsona.

Model ten może być trenowany poprzez uczenie maszynowe, czyli proces, w którym komputer samodzielnie uczy się na podstawie danych i doświadczenia. W przypadku rozpoznawania choroby Parkinsona na podstawie próbek głosu. Jakość takiego modelu zależy od tego jak przygotujemy dane, jakich narzędzi związanych z uczeniem maszynowym wykorzystamy oraz ile jest danych uczącyc i treningowych. Naprzemienne korzystanie z różnych narzędzi w różnych konfiguracjach pozwoli nie tylko poprawić jakość modelu, ale także lepiej zrozumieć dane oraz sam problem.

Gdy model jest gotowy, może być używany do przewidywania prawdo-

podobieństwa choroby, możne zostać wtedy wykorzystany przez lekarzy do wczesnego rozpoznania choroby i leczenia pacjentów.

Rozdział 2

Opis metody

2.1 Wprowadzenie teoretyczne

2.1.1 Regresja logistyczna

Regresja logistyczna jest metodą statystyczną, która służy do przewidywania prawdopodobieństwa danego wyniku na podstawie danych wejściowych. Jest często używana do rozpoznawania chorób lub predykcji innych zdarzeń binarnych. Regresja logistyczna opiera się na założeniu, że prawdopodobieństwo danego wyniku zależy od pewnych cech wejściowych.

Wzór na regresję logistyczną

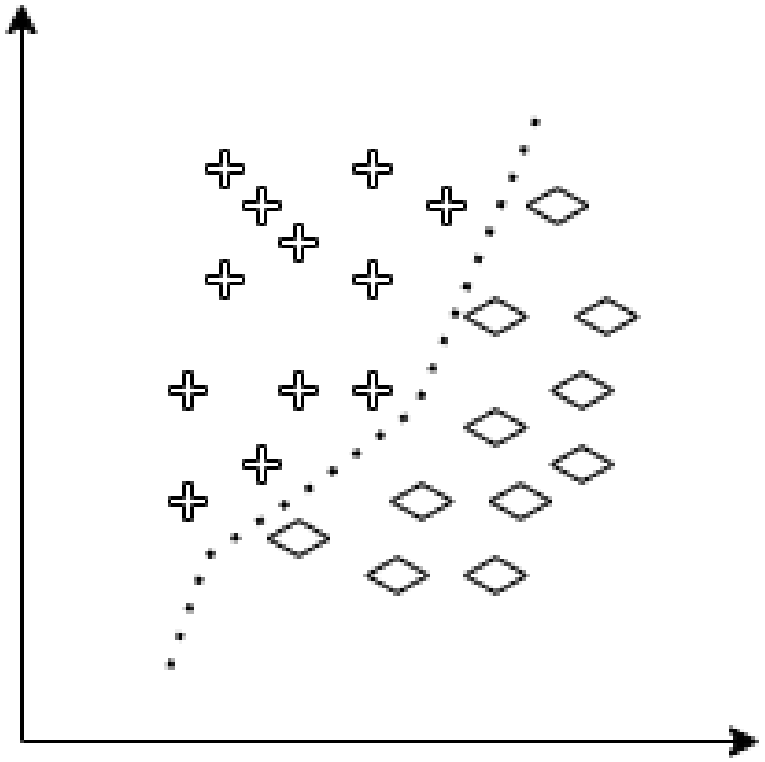
$$P(Y = 1|x_1, x_2, \ldots, x_k) = \frac{e^{a_0 + \sum_{i=1}^{k} a_i x_i}}{1 + e^{a_o + \sum_{i=1}^{k} a_i x_i}}$$

Symbole:

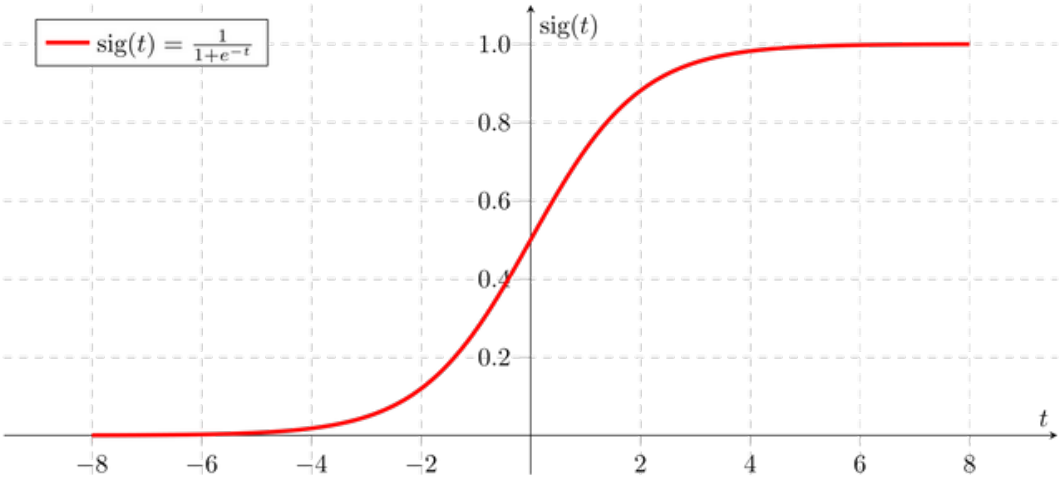
P(Y=1—x_1,x_2,...,x_k) - warunkowe prawdopodobieństwo, że zmienna zależna Y przyjmie wartość równą 1 dla wartości zmiennych niezależnych x_1,x_2,...,x_n
e - liczba Eulera
a_0 - stała
a_1,a_2,...,a_k - współczynniki regresji dla poszczególnych zmiennych niezależnych, predyktorów
x_1,x_2,...,x_k - zmienne niezależne, predyktory, zmienne wyjaśniające.

Model regresji logistycznej ma za zadanie znaleźć taką funkcję, która od-

dzieliłaby dwa (lub więcej) zbiory, tak aby nad nią były wartości dla jednej etykiety, a poniżej wartości drugiej, dzięki temu, po wprowadzeniu podobnie przygotowanych danych model będzie w stanie określić jak zaklasyfikować dany obiekt.



Jak w poprzednim wzorze można zobaczyć, dzieje się to na bazie prawdopodobieństwa. Naszą funkcję $\theta^T X$ zamkniemy w funkcji sigmoidalnej:



gdzie t, jest wartością naszej funkcji. Jak widać, funkcja ta przyjmuje war-

tości z przedziału (0, 1), 0 przyjmuje dla $-\infty$, a wartość 1 dla ∞. Dla zera, wartość funkcji jest równa 0,5. Oznacza to, że prawdopodobieństwo tego, że obiekt, któremu przypisujemy etykiete ma etykietę Y=1, jest równe:

$$P(Y = 1|x; \theta) \ = \ sig\left(\theta^T X\right) \ = \ h_\theta(x).$$

Dla zera wzór wygląda następująco:

$$P(Y = 0|x; \theta) \ = 1 - \ h_\theta(x).$$

Zatem ogólny wzór na prawdopodobieństwo prezentuje się w taki sposób:

$$P(y|x; \theta) \ = h_\theta(x)^y (1 - h_\theta(x))^{1-y}$$

Dla zbioru wszystich cech X:

$$L(\vec{y}|X; \theta) \ = \ \prod_{i \ = \ 1}^{n} h_\theta\left(x^i\right)^{y^i} \left(1 - h_\theta\left(x^i\right)\right)^{1-y^i}$$

Celem wyuczenia modelu jest zmaksymalizowanie tej funkcji względem parametru θ. Oznacza to, że należy znaleźć pochodną tej funkcji po θ. Na szczęście pochodna fukncji sigmoidalnej wynosi
$sig'(t) \ = \ sig(t)(1 - sig(t))$.
Końcowy algorytm wygląda więc w następujący sposób:

$$\theta \leftarrow \theta + \eta \, \Sigma_{i=1}^{m} \left(y^i - h_\theta\left(x^i\right)\right) x^i$$

2.1.2 LogisticRegression

Jest to klasa z biblioteki scikit-learn (sklearn), która implementuje regresję logistyczną. Posiada ona gotową metodę fit(), która trenuje model na podanych danych wejściowych. Po wytrenowaniu można użyć metody predict() do przewidywania klas dla nowych przypadków na podstawie wytrenowanego modelu oraz predict_proba(), która pozwala sprawdzić jakie jest prawdopodobieństwo dla jakiej etykiety dla nowego przypadku. Klasa LogisticRegression oferuje opcję regularyzacji, która może pomóc zapobiec nadmiernemu dopasowaniu (tzw. "przeuczeniu") modelu do danych treningowych. W tym projekcie korzystamy mi. z L2, zwanej również Lasso.

2.1.3 RFE

RFE (ang. Recursive Feature Elimination) to metoda regularyzacji w uczeniu maszynowym, która służy do wybierania najważniejszych cech danych do użycia w modelu uczenia maszynowego. Jest ona często stosowana w

przypadku, gdy dane wejściowe zawierają wiele cech, a chcemy wybrać te, które najlepiej wpływają na dokładność modelu.

RFE działa poprzez rekurencyjne usuwanie cech z danych wejściowych i sprawdzanie, jak to wpływa na dokładność modelu. Najpierw wybierane są najmniej ważne cechy, a następnie usuwane są kolejne, aż do momentu, gdy wybrane cechy dają najlepszą dokładność.

2.1.4 Polynomial features

Polynomial features to metoda polegająca na dodawaniu nowych cech do danych wejściowych, które są kombinacjami istniejących już cech w kolejnych potęgach. Na przykład, jeśli mamy dwa cechy wejściowe x_1 i x_2, polynomial features pozwolą nam dodać do danych nowe cechy, takie jak x_1^2, x_1^3, x_2^2, $x2_3$, itd. Polynomial features mogą być używane w regresji logistycznej w celu zwiększenia dokładności modelu poprzez uwzględnienie zależności między cechami wejściowymi.

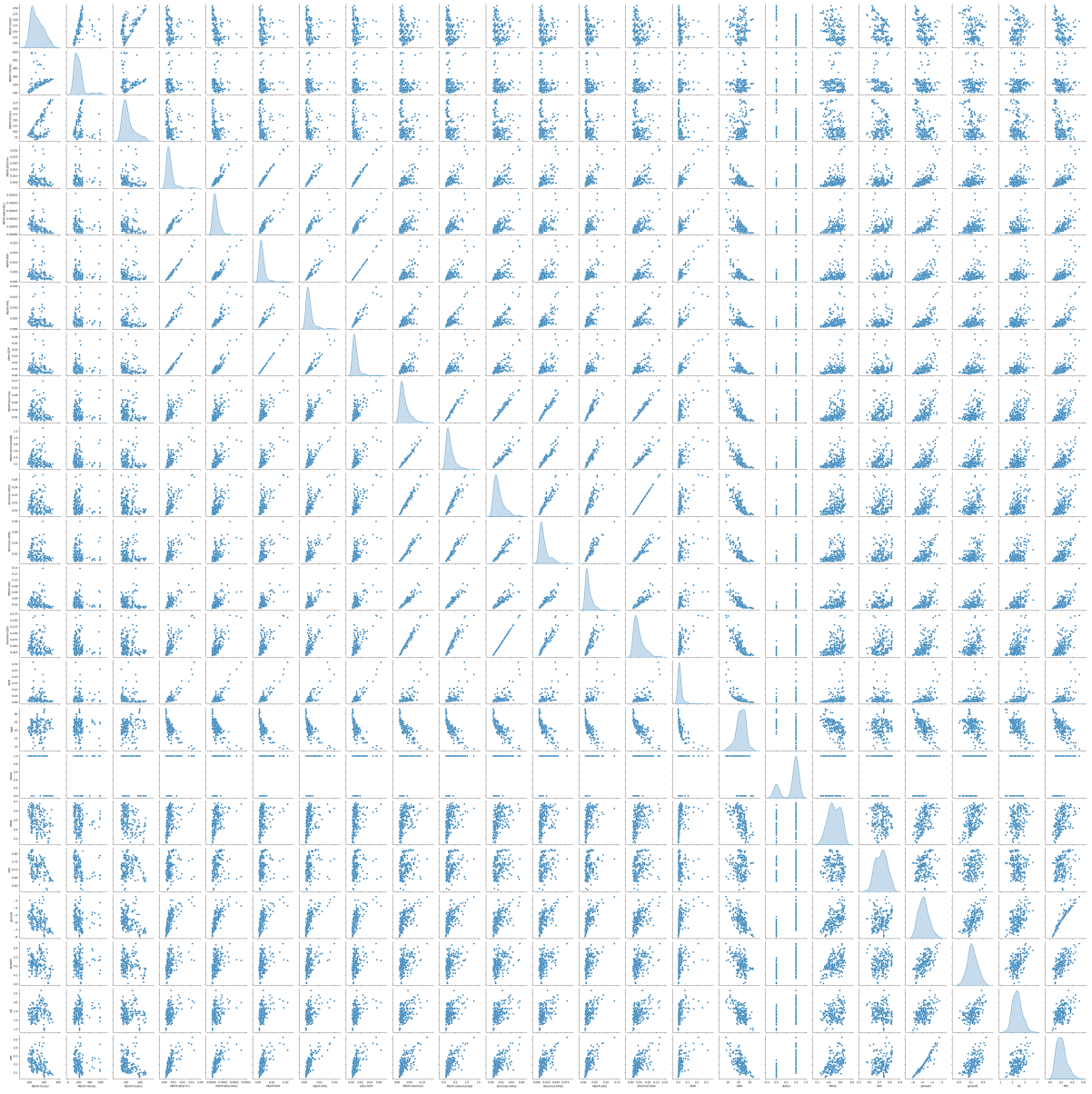
2.1.5 Analiza danych wejściowych

Dane wejściowe, na których będziemy badać model pochodzą ze strony `https://archive.ics.uci.edu/ml/datasets/parkinsons` i zawierają 195 krótkich nagrań głosowych od pacjentów badania choroby Parkinsona, każde nagranie posiada 24 cechy. Nagrania w tym zbiorze pochodzą od 31 osób, z czego 23 z nich chorych. Oznacza to, że na każdego pacjenta przypada około 6 nagrań. Do atrybutów każdego nagrania należą:

- Nazwa pacjenta.
- Średnia, maksymalna i minimalna częstotliwość głosu.
- Drganie głosu (kilka atrybutów) - krótkookresowe odchylenie od ustalonych, okresowych charakterystyk sygnału.
- Szmer (kilka atrybutów)
- NHR, HNR - współczynniki noise/harmonic ratio i harmonic/noise ratio.
- Status - określa czy pacjent jest zdrowy, czy chory, będziemy próbować nauczyć nasz model przewidywania tej wartości.
- RPDE, D2
- DFA - W procesach stochastycznych, teorii chaosu i analizie szeregów czasowych analiza wahań detrendowanych jest metodą określania statystycznego samopowinowactwa sygnału.

- spread1, spread2, PPE - trzy nieliniowe wartości określające wariancję bazowej częstotliwości.

Jak widać, badania zostały przeprowadzone w bardzo małej skali. Mało danych wejściowych może wpłynąć na jakość naszego modelu. Cech również nie jest dużo. Niewielka ilość danych i cech wpływa jednak pozytywnie na czas, w jaki model zostanie nauczony. Dzięki temu, możemy nasz model poddać większej ilości prób i dobrać odpowiednie parametry i techniki do stworzenia jak najlepszego modelu.
Poniższe wykresy przedstawiają, jak poszczególne cechy nagrań mają się do siebie, czy są skorelowane, czy nie.



Dane z końca wyglądają następująco:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jit
190	phon_R01_S50_2	174.188	230.978	94.261	0.00459	0.00003	0.00263	0.00259	
191	phon_R01_S50_3	209.516	253.017	89.488	0.00564	0.00003	0.00331	0.00292	
192	phon_R01_S50_4	174.688	240.005	74.287	0.01360	0.00008	0.00624	0.00564	
193	phon_R01_S50_5	198.764	396.961	74.904	0.00740	0.00004	0.00370	0.00390	
194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	0.00003	0.00295	0.00317	

5 rows × 24 columns

Jak widać, kolumna name zawiera łańcuchy znaków, oznaczają one od której osoby pochodzą nagrania. Z tych dwóch powodów nie bierzemy je pod uwagę.

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	1
mean	154.228641	197.104918	116.324631	0.006220	0.000044	0.003306	0.003446	0.009920	
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.002759	0.008903	
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.000920	0.002040	
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.001860	0.004985	
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.002690	0.007490	
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.003955	0.011505	
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.019580	0.064330	

8 rows × 23 columns

Z poprzednich dwóch obrazków wynika, że dane nie są ustandaryzowane, jest to jedna z pierwszych rzeczy, przez którą jakość modelu może się pogorszyć. Dlatego zostanie ona wyeliminowana poprzez standaryzację.
Ze względu na niewielką ilość danych, podział danych na zbiór treningowy zostanie wykonany z słabszą nierównością jednego zbioru względem drugiego. Za mało danych testowych mogłoby niekorzystnie wpłynąć na ocenę modelu. Należy również ręcznie wyciąć kolumnę, trzymającą wynik, tak sporządzone dane są gotowe do badania.

2.2 Badania symulacyjne

2.2.1 Przygotowanie danych

Po wyciągnięciu wniosków z analizy danych usunęliśmy kolumnę 'name' oraz wydzieliliśmy zbiór na data i target. Po tym wszystkim pozostała nam jeszcze wcześniej wspomniana standaryzacja. Dane zostały wyskalowane dzięki

klasie StandardScaller pochodzącej z bilbioteki sklearn. Tak prezentują się dane po przeskalowaniu:

	MDVP:Fo	MDVP:Fhi	MDVP:Flo	MDVP:Jitter	MDVP:Jitter	MDVP:RAP	MDVP:PPQ	Jitter:DDP	M
count	1.950000e+02	1.950000e+02	1.950000e+02	1.950000e+02	1.950000e+02	1.950000e+02	1.950000e+02	1.950000e+02	
mean	3.529940e-17	-2.237526e-16	1.309494e-16	-3.586874e-17	1.024821e-16	-1.182103e-16	9.351494e-17	8.234154e-17	
std	1.002574e+00	1.002574e+00	1.002574e+00	1.002574e+00	1.002574e+00	1.002574e+00	1.002574e+00	1.002574e+00	
min	-1.596162e+00	-1.040581e+00	-1.171366e+00	-9.389487e-01	-1.064103e+00	-8.872543e-01	-9.180440e-01	-8.873331e-01	
25%	-8.879183e-01	-6.820590e-01	-7.379376e-01	-5.708520e-01	-6.898141e-01	-5.561906e-01	-5.764609e-01	-5.557071e-01	
50%	-1.317379e-01	-2.331437e-01	-2.766579e-01	-2.647942e-01	-4.018994e-01	-2.724216e-01	-2.748504e-01	-2.736279e-01	
75%	6.913210e-01	2.969710e-01	5.458200e-01	2.366858e-01	4.618447e-01	1.785683e-01	1.848331e-01	1.784870e-01	
max	2.564598e+00	4.327631e+00	2.829908e+00	5.570985e+00	6.220139e+00	6.125892e+00	5.862742e+00	6.126923e+00	

8 rows × 22 columns

Jak widać standardowe odchylenie jest równe 1, a średnia w danych jest zbliżona do zera. Taki zabieg nie zmienia topologii danych, wyśrodkowuje je względem punktu przecięcia wszystkich osi.

```
First patient in database
[-0.77097169 -0.53097409 -0.05772056  0.71541825  1.03767418  0.45389169
  1.27680862  0.45268371  1.68173116  1.7684643   1.54791153  2.27650439
  1.15945391  1.54825419 -0.13784252 -0.63450828 -0.38752443  1.83756192
  1.47985265  1.31118546  0.27507712  1.80360503]
---Mean---
[ 2.96059473e-17 -2.27738056e-16  1.04759506e-16  1.82190445e-17
  9.10952225e-17 -1.27533312e-16  6.37666558e-17  7.28761780e-17
  2.36847579e-16 -1.63971401e-16  1.06467541e-16  2.62468110e-16
 -6.83214169e-17 -1.69380179e-16  1.09314267e-16  8.19857003e-16
 -1.09314267e-16  6.55885602e-16  1.20245694e-15 -1.13869028e-16
 -5.92118946e-16  1.36642834e-17]
--std---
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Przeskalowanie wartości widać na pierwszym nagraniu.
Ostatnim krokiem w przygotowaniu danych jest podział danych na zbiór uczący i testowy. Ze względu na małą ilość danych, zbiór testowy będzie stanowił 20% całości.

```
Training dataset:
patients_train_data: (156, 22)
patients_train_target: (156,)
Testing dataset:
patients_test_data: (39, 22)
patients_test_target: (39,)
```

2.2.2 Analiza jakości uzyskanego narzędzia

a) regresja logistyczna z domyslnymi wartościami

```
from sklearn.metrics import accuracy_score
acc = accuracy_score(parkinson_test_target, logistic_regression.predict(parkinson_test_data))
print("Model accuracy is {0:0.2f}".format(acc))
```

```
Model accuracy is 0.77
```

```
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(parkinson_test_target, logistic_regression.predict(parkinson_test_data))
print(conf_matrix)
```

```
[[ 7  3]
 [ 6 23]]
```

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(LogisticRegression(), parkinson_scaled, parkinson_target, cv=4)
print(scores)
```

```
[0.91836735 0.87755102 0.69387755 0.75      ]
```

Wynik accuracy score wynosi 0.77, co oznacza, że model dobrze przewiduje 77 procent przypadków.

Analizując macierz konfuzji mozna dojść do wniosku, że model poprawnie zaklasyfikował 7 przypadków z klasy 0 i 23 przypadki z klasy 1. Natomiast błędnie zaklasyfikował 3 przypadki z klasy 0 jako klasę 1 oraz 6 przypadków z klasy 1 jako klasę 0.

Wyniki dla metody cross validation są dość zróżnicowane, co może wskazywać na to, że model nie jest do końca stabilny i jego jakość może się znacznie różnić w zależności od zbioru danych, może to oznaczać, że zbiór danych jest w pewnym (niewielkim) stopniu wadliwy.

Ogólnie rzecz biorąc, model uzyskał dobrą skuteczność, ale nie jest on idealny. Może być potrzebna dalsza optymalizacja modelu lub użycie innego rodzaju modelu, aby uzyskać lepsze wyniki.

b) użycie polynomial features dla regresji logistycznej z domyślnymi wartościami

```
acc_p = accuracy_score(parkinson_test_target, poly_regression.predict(parkinson_test_poly))
print("Model accuracy with poly features is {0:0.2f}".format(acc_p))
```

```
Model accuracy with poly features is 0.90
```

```
conf_matrix = confusion_matrix(parkinson_test_target, poly_regression.predict(parkinson_test_poly))
print(conf_matrix)
```

```
[[10  0]
 [ 4 25]]
```

```
scores = cross_val_score(LogisticRegression(), parkinson_scaled_poly, parkinson_target, cv=4)
print(scores)
```

```
[0.89795918 0.7755102  0.85714286 0.77083333]
```

Wynik accuracy score wynosi 0.90, co oznacza, że model dobrze przewiduje 90 procent przypadków.

Analizując macierz konfuzji mozna dojść do wniosku, że model poprawnie zaklasyfikował 10 przypadków z klasy 0 i 25 przypadki z klasy 1. Nato-

miast błędnie zaklasyfikował 4 przypadki z klasy 1 jako klasę 0.

Wyniki dla metody cross validation są dość zbliżone do siebie, co może wskazywać na to, że model jest bardziej stabilny niż w poprzednim przypadku.

Jeśli porównamy te wyniki z poprzednimi, to możemy zauważyć, że model z polynomial features uzyskał wyższy wynik accuracy score oraz lepsze wyniki w macierzy konfuzji. Cross validation również pokazuje lepsze wyniki niż w poprzednim przypadku. Wszystko to wskazuje na to, że model z polynomial features jest lepszy od modelu bez tych cech.

c) użycie RFE
RFE dla polynmial features

```
from sklearn.feature_selection import RFE


for i in range(10, 140, 20):
    sel_ = RFE(estimator=LogisticRegression(), n_features_to_select=i)
    sel_.fit(parkinson_train_poly, parkinson_train_target)
    print("for i = ", i, ", score = ", accuracy_score(parkinson_test_target, sel_.predict(parkinson_test_poly)))
```

```
for i =  10 , score =  0.8717948717948718
for i =  30 , score =  0.8717948717948718
for i =  50 , score =  0.8974358974358975
for i =  70 , score =  0.8974358974358975
for i =  90 , score =  0.8974358974358975
for i =  110 , score =  0.8974358974358975
for i =  130 , score =  0.8974358974358975
```

RFE dla samej regresji logistycznej

```
for i in range(1, 20, 1):
    sel_ = RFE(estimator=LogisticRegression(), n_features_to_select=i)
    sel_.fit(parkinson_train_data, parkinson_train_target)
    print("for i = ", i, ", score = ", accuracy_score(parkinson_test_target, sel_.predict(parkinson_test_data)))
```

```
for i =  1 , score =  0.7948717948717948
for i =  2 , score =  0.7435897435897436
for i =  3 , score =  0.7948717948717948
for i =  4 , score =  0.7692307692307693
for i =  5 , score =  0.8205128205128205
for i =  6 , score =  0.8205128205128205
for i =  7 , score =  0.7692307692307693
for i =  8 , score =  0.8461538461538461
for i =  9 , score =  0.7692307692307693
for i =  10 , score =  0.7692307692307693
for i =  11 , score =  0.7692307692307693
for i =  12 , score =  0.7692307692307693
for i =  13 , score =  0.7692307692307693
for i =  14 , score =  0.7948717948717948
for i =  15 , score =  0.7948717948717948
for i =  16 , score =  0.7948717948717948
for i =  17 , score =  0.7948717948717948
for i =  18 , score =  0.7948717948717948
for i =  19 , score =  0.7692307692307693
```

```
for i in range(1, 20, 1):
    scores = cross_val_score(RFE(estimator=LogisticRegression(), n_features_to_select=i), parkinson_scaled, parkinson_target, cv=
    print(scores)
```

```
[0.91836735 0.91836735 0.69387755 0.79166667]
[0.89795918 0.81632653 0.69387755 0.77083333]
[0.89795918 0.85714286 0.65306122 0.79166667]
[0.87755102 0.85714286 0.69387755 0.8125    ]
[0.89795918 0.87755102 0.71428571 0.77083333]
[0.89795918 0.87755102 0.71428571 0.77083333]
[0.91836735 0.87755102 0.71428571 0.75      ]
[0.91836735 0.87755102 0.75510204 0.75      ]
[0.91836735 0.87755102 0.73469388 0.75      ]
[0.91836735 0.87755102 0.73469388 0.75      ]
[0.89795918 0.87755102 0.75510204 0.75      ]
[0.89795918 0.87755102 0.7755102  0.75      ]
[0.89795918 0.87755102 0.7755102  0.75      ]
[0.89795918 0.87755102 0.71428571 0.75      ]
[0.89795918 0.87755102 0.69387755 0.75      ]
[0.89795918 0.87755102 0.69387755 0.75      ]
[0.91836735 0.87755102 0.69387755 0.75      ]
[0.91836735 0.87755102 0.69387755 0.75      ]
[0.91836735 0.87755102 0.69387755 0.75      ]
```

Jak można zauważyć najlepsze wyniki zostały uzyskane dla modelu, gdzie zastosowano polynomial features oraz liczba cech była większa lub równa 50. Fakt, że wynik nie pogarsza się wraz ze wzrostem cech może oznaczać ich takie ułożenie, że model nie przeucza się. Najlepsze wyniki bez polynomial features zostały uzyskane dla wartości cech równej 8. Nie są one jednak lepsze niż te gdzie zostały uzyte polynomial features.

d)regresja logistyczna, na której zostało użyte Grid Search

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
parameters = {'penalty': ['l1', 'l2'], 'C': [0.1, 1, 10]}

logistic_regression = LogisticRegression()

grid_search = GridSearchCV(logistic_regression, parameters)
grid_search.fit(parkinson_train_data, parkinson_train_target)
best_params = grid_search.best_params_
print("Najlepsze parametry: ", best_params)
```

```
Najlepsze parametry:  {'C': 0.1, 'penalty': 'l2'}
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

logistic_regression = LogisticRegression(penalty='l2', C=0.1)
logistic_regression.fit(parkinson_train_data, parkinson_train_target)

acc = accuracy_score(parkinson_test_target, logistic_regression.predict(parkinson_test_data))
print("Model accuracy is {0:0.2f}".format(acc))

conf_matrix = confusion_matrix(parkinson_test_target, logistic_regression.predict(parkinson_test_data))
print(conf_matrix)

scores = cross_val_score(LogisticRegression(penalty='l2', C=0.1), parkinson_scaled, parkinson_target, cv=4)
print(scores)
```

```
Model accuracy is 0.85
[[ 6  4]
 [ 2 27]]
[0.93877551 0.87755102 0.73469388 0.75      ]
```

Maroda Grid Search pokazała, że najlepszymi parametrami dla regresji logi-

stycznej dla wprowadzonych danych będą penalty = l2 oraz C = 0.1 Analizując wyniki można zauważyć sporą poprawę względem wyników uzyskanych dla pierwszego wyniku (samej regresji logistycznej)

e)Polynomial features z regresją logistyczną uzywającą Grid Search

```
poly_regression = LogisticRegression(penalty='l2', C=0.1)
poly_regression.fit(parkinson_train_poly, parkinson_train_target)
```

```
LogisticRegression(C=0.1)
```

```
acc_p = accuracy_score(parkinson_test_target, poly_regression.predict(parkinson_test_poly))
print("Model accuracy with poly features is {0:0.2f}".format(acc_p))
```

```
Model accuracy with poly features is 0.95
```

```
conf_matrix = confusion_matrix(parkinson_test_target, poly_regression.predict(parkinson_test_poly))
print(conf_matrix)
```

```
[[ 8  2]
 [ 0 29]]
```

```
scores = cross_val_score(LogisticRegression(penalty='l2', C=0.1), parkinson_scaled_poly, parkinson_target, cv=4)
print(scores)
```

```
[0.91836735 0.79591837 0.83673469 0.75      ]
```

Analizując wyniki można zauważyć, że jest to dotychczasowny najlepszy model. Jego wyniki są bardzo wysokie, w szczególności acccuracy score, kótre wynosi 0.95.

f)RFE z Grid Search
Nie ma potrzeby sprawdzać jak RFE wpłynie na model po poprzednim wykorzystaniu Grid Search, gdyż wybrał on metodę L2, która już jest metodą regularyzacji.

Rozdział 3

Podsumowanie

3.1 Wnioski

Jak można zauważyć, uczenie maszynowe, a konkretnie regresja logistyczna zaskakująco dobrze potrafi sobie poradzić z rozpoznawaniem choroby Parkinsona nawet na tak niewielkim zbiorze danych. Lepsze wyniki i zrozumienie problemy mogłyby być osiągnięte, gdyby ten zbiór był większy i bardziej zróżnicowany. Zbiór nagrań od tylko 31 pacjentów, których jest 195, wydaje się być bardzo ograniczony. Do stworzenia lepszego modelu idealnie by było posiadać większą ilość nagrań od większej ilości pacjentów, gdyż obecna liczba wydaje się być ograniczająca. Jednak mimo tego model osiąga dość wysokie wyniki. Analizie zostało poddane uzycie regresji logistycznej z oraz bez polynomial features, RFE oraz Grid Search. Sprawdzając kolejne wyniki można zauważyć jak poprawia się jakość modelu, aż do jego najlepszej uzyskanej wersji, czyli regresji liniowej z parametrem c=0.1 oraz polynomial features. Użycie tych wszytskich narzędzi było jednocześnie najbardziej czasochłonne porównując do pozostałych prób. Czas tworzenia takiego modelu nie jest jednak i tak długi, ponieważ trwał około minuty, co w uczeniu maszynowym jest bardzo krótkim okresem uczenia modelu.

Dodatek A

Kod programu

https://github.com/tzapadlinski/KWD_Projekt