



Politechnika Łódzka

Wydział Fizyki Technicznej, Informatyki
i Matematyki Stosowanej

Praca końcowa

Klasyfikacja bólów głowy z wykorzystaniem algorytmów uczenia maszynowego*

Konrad Pławik

Promotor:
dr hab. inż. Agnieszka Wosiak

Czerwiec 2024

* SVN: <https://github.com/kplawik/HeadacheClassification>

Spis treści

1. Wstęp	3
2. Dane	4
2.1. Zbiór danych	4
2.2. Informacje prawne	4
3. Wstęp teoretyczny	6
3.1. Klasyfikacja	6
3.1.1. Klasyfikator kNN (k-najbliższych sąsiadów)	6
3.1.2. Naiwny Klasyfikator Bayesa	6
3.1.3. SVM - Maszyna Wektorów Nośnych	7
3.2. Ewaluacja modelu klasyfikacyjnego	8
3.2.1. Macierz pomyłek	8
3.2.2. Walidacja krzyżowa	8
3.2.3. Pozostałe podstawowe metryki	8
3.3. Sztuczne sieci neuronowe	8
3.3.1. Perceptron wielowarstwowy	8
3.3.2. Sieci głębokie	9
3.3.3. Funkcja aktywacji softmax	9
4. Techniki SHAP i LIME	10
4.1. SHAP	10
4.2. LIME (Local Interpretable Model-agnostic Explanations)	10
4.2.1. LIME dla danych tabelarycznych i klasyfikacji wieloklasowej	11
5. Eksperymenty	13
5.1. Klasyfikacja kNN przy podziale zbioru 80:20	13
5.1.1. Wyniki	13
5.2. Klasyfikacja kNN przy podziale zbioru 75:25	13
5.2.1. Wyniki	14
5.3. Klasyfikacja Naiwnym Klasyfikatorem Bayesa	14
5.3.1. Wyniki	15
5.4. Selekcja i ekstrakcja cech	16
5.4.1. Wyniki	17
5.5. Perceptron wielowarstwowy	17
5.5.1. Wyniki	18
5.6. Sieć głęboka z funkcją aktywacji Softmax	19
5.6.1. Wyniki	20
Spis rysunków	22
Literatura	22

1. Wstęp

Bóle głowy bywają trudne do sklasyfikowania. O ile z obserwacji własnych miałem niestety okazję się o tym przekonać to nawet i świat nauki od lat również boryka się z tym problemem. Brytyjski instytut znany jako Headache Classification Committee of the International Headache Society (IHS) rozróżnia 13 kategorii bólów głowy - a samej tylko migreny - 29 typów [1]. Co więcej instytut ten wyraźnie mówi o tym że pacjent może cierpieć na więcej niż jeden z rodzaj ([1] punkt 9 we wstępie). Badania przeprowadzone przez EHF (European Headache Federation) [2] również potwierdzają że dominujący ból głowy nie musi być jedynym [3].

W pomocą przychodzi nam zagadnienie Uczenia Maszynowego oraz powiązane z nim algorytmy klasyfikacyjne. Poniższa praca dokumentuje wyniki kilkudziesięciu eksperymentów mających na celu automatyczną klasyfikację przy użyciu zarówno algorytmów regresyjnych (np. kNN) jak i głębokich Sieci Neuronowych (Deep Learning).

2. Dane

2.1. Zbiór danych

Wykorzystany zbiór danych pozyskano z serwisu `codeocean.com` [4]. Zbiór ten udostępniona na licencji GNU General Public License (GPL) a jego autorami są:

1. Paola A. Sánchez-Sánchez
2. José Rafael García-González
3. Juan Manuel Rua Ascar.

Cała trójka z pochodzi Universidad Simón Bolívar, Barranquilla w Kolumbii.

Zbiór zawierał anonimowe dane 400 rozpoznanych przypadków a każdy z przypadków 23 cechy. Cechy miały różny typ (np. wiek pacjenta (typ całkowity) czy wystąpienie danego objawu (typ binarny)) co przemawiało za użyciem normalizacji przy użyciu MinMaxScalera z biblioteki Scikit-learn.

W zbiorze znajdowały się dane dotyczące 7 rodzajów bólu głowy. Zbiór nie był zbiorem zbalansowanym (co należy mieć na uwadze w dalszej analizie):

Type	
Basilar-type aura	18
Familial hemiplegic migraine	24
Migraine without aura	60
Other	17
Sporadic hemiplegic migraine	14
Typical aura with migraine	247
Typical aura without migraine	20
dtype:	int64

Zbiór nie posiadał brakujących danych więc nie zaistniała konieczność imputacji.

2.2. Informacje prawne

Zbiór udostępniony został na licencji GNU General Public License (GPL) [4].

Wykorzystane oprogramowanie korzystało z licencji:

- Język Python: Python Software Foundation License [5]
- Biblioteka Pandas: BSD 3-Clause License [6]
- Biblioteka NumPy: BSD 3-Clause License [7]
- Biblioteka Seaborn: BSD 3-Clause License [8]
- Biblioteka TensorFlow: Apache License 2.0 [9]

Wspomniane biblioteki zostały szczegółowo opisane w następujących publikacjach naukowych:

- Język Python [10]
- Pandas: <https://zenodo.org/records/10957263> [11]
- NumPy: <https://www.nature.com/articles/s41586-020-2649-2> [12]

Seaborn: <https://joss.theoj.org/papers/10.21105/joss.03021> [13]
TensorFlow: <https://zenodo.org/records/10798587> [14]

3. Wstęp teoretyczny

3.1. Klasyfikacja

3.1.1. Klasyfikator kNN (k-najbliższych sąsiadów)

Klasyfikator kNN [15], ze względu na swoją intuicyjność, jest jednym z najpopularniejszych klasyfikatorów. Działa on zgodnie z regułą: obserwacja x zostaje sklasyfikowana do najliczniejszej klasy z pośród k obserwacji najbliższych punktowi x .

Szacowane prawdopodobieństwo przynależności obserwacji x do danej klasy wśród x najbliższych sąsiadów, zapisujemy jako:

$$\hat{j}|\mathbf{x} = \frac{1}{k} \sum_{i=1}^n l(\rho(\mathbf{x}, \mathbf{x}_i) \leq \rho(\mathbf{x}, \mathbf{x}^{(k)})) l(y_i = j), \quad j = 1, \dots, g \quad (1)$$

gdzie:

$x^{(k)}$ - jest k -tym co do odległości x punktem z próby uczącej

ρ - jest pewną odległością, określaną jako miara niepodobieństwa.

3.1.2. Naiwny Klasyfikator Bayesa

Jest klasyfikatorem probabilistycznym [15], który opiera się na użyciu twierdzenia.

$$P(C|F_1, \dots, F_n) \quad (2)$$

gdzie:

C - oznacza zmienną zależną, będącą zbiorem etykiet klas

F_1, \dots, F_n - cechami opisującymi zbiór przypadków.

Korzystając z twierdzenia Bayesa, które mówi że dla dowolnej hipotezy $h \in \mathcal{H}$ oraz zbioru danych D zachodzi równość:

$$P(h|D) = \frac{P(h)P(D|h)}{P(D)} \quad (3)$$

oraz niezależności warunkowej cech F_i oraz F_j dla $i \neq j$ otrzymujemy:

$$P(F_i|C, F_j) = P(F_i|C) \quad (4)$$

Co ostatecznie daje:

$$P(C|F_1, \dots, F_n) = \frac{1}{Z} P(C) \prod_{i=1}^n P(F_i|C) \quad (5)$$

przy czym Z oznacza współczynnik skalujący, zależny od atrybutów F_1, \dots, F_n i ustalony w przypadku znanych wartości atrybutów cech.

Zatem wynik działania naiwnego klasyfikatora bayesowskiego można zapisać jako:

$$\text{klasa}(f_1, \dots, f_n) = \arg \max_c P(C = c) \prod_{i=1}^n P(F_i = f_i | C = c) \quad (6)$$

3.1.3. SVM - Maszyna Wektorów Nośnych

Maszyna Wektorów Nośnych (Maszyna Wektorów Wspierających) [15] jest to abstrakcyjny koncept maszyny, która działa jak klasyfikator, a której nauka ma na celu wyznaczenie hiperpłaszczyzny rozdzielającej z maksymalnym marginesem przykłady należące do dwóch klas. [16]

Zadanie sprowadza się do znalezienia granicy decyzyjnej między klasami i związane jest z pojęciem separowalności liniowej, zgodnie z którym dwie klasy są liniowo separowalne, gdy istnieje hiperpłaszczyzna H postaci $g(x)$ wyrażona równaniem:

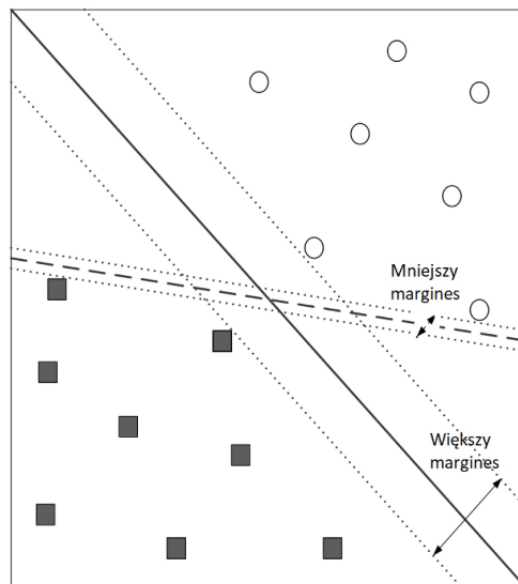
$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b. \quad (7)$$

przyjmująca wartości:

$$\begin{cases} g(\mathbf{x}_i) > 0 & \text{jeśli } \mathbf{x}_i \in 1, \\ g(\mathbf{x}_i) < 0 & \text{jeśli } \mathbf{x}_i \in -1 \end{cases} \quad (8)$$

gdzie: \mathbf{x} - oznacza wektor danych, zaś \mathbf{w} oraz b są parametrami modelu.

W rezultacie można uzyskać zbiór wielu możliwych rozwiązań (czyli hiperpłaszczyzn), z których wybierane jest takie, które maksymalizuje margines klasyfikatora liniowego:



Rysunek 1. Schemat działania SVM

Powyższy rysunek przedstawia uproszczony schemat działania SVM - źródło: materiały wykładowe.

3.2. Ewaluacja modelu klasyfikacyjnego

3.2.1. Macierz pomyłek

Macierz pomyłek (błędów, ang. confusion matrix) [15], inaczej określana mianem tablicy kontyngencji (ang. contingency table). prezentuje liczby przypadków należących do poszczególnych poprawnych klas decyzyjnych oraz tych, które są przewidywane.

W przypadku wielu etykiet macierz pomyłek jest macierzą kwadratową $m \times m$:

	Class ₁	Class ₂	...	Class _m
Class ₁	n_{11}	n_{12}	...	n_{1m}
Class ₂	n_{21}	n_{22}	...	n_{2m}
\vdots	\vdots	\vdots	\vdots	\vdots
Class _m	n_{m1}	n_{m2}	...	n_{mm}

W klasyfikacji wieloklasowej oznaczamy często tylko jedną klasę jako pozytywną, a pozostałe łącznie definiujemy jako negatywne - sprowadzając problem do wielu klasyfikacji binarnych.

3.2.2. Walidacja krzyżowa

W związku z tym że wykorzystany wzór nie jest zbiorem zbyt licznych wartości zastosować tzn. walidację krzyżową (inaczej kroswalidację, ang. *cross-validation*).

1. K-krotna walidacja krzyżowa polega na podzieleniu całego zbioru przypadków na k rozłącznych i równolicznych części
2. Kolejno każda z tych części stanowi wydzielony zbiór testowy, a pozostałe $k-1$ - zbiór uczący.
3. Walidację przeprowadza się na każdej z k części zbioru, zaś wynik końcowy jest średnią z poszczególnych wyników częściowych.

3.2.3. Pozostałe podstawowe metryki

W pracy wykorzystano również: dokładność (*Accuracy*), precyzja (*Precision*), czułość (*Sensitivity*), specyficzność (*Specificity*) oraz predykcję klasy negatywnej (*Negative Predictive Value*).

3.3. Sztuczne sieci neuronowe

3.3.1. Perceptron wielowarstwowy

Perceptron wielowarstwowy (MLP, ang. *Multi-Layer Perceptron*) jest podstawowym typem sztucznej sieci neuronowej, który składa się z co najmniej trzech warstw: warstwy wejściowej, jednej lub więcej warstw ukrytych oraz warstwy wyjściowej. Każdy neuron w jednej warstwie jest połączony z każdym neuronem w warstwie następnej, co umożliwia przetwarzanie skomplikowanych wzorców i zależności. W przeciwieństwie do prostego perceptronu jednopoziomowego, MLP jest zdolny do rozwiązywania problemów, które

nie są liniowo separowalne. Proces uczenia w MLP opiera się na algorytmie wstecznej propagacji błędów, który minimalizuje błąd sieci poprzez dostosowanie wag połączeń między neuronami. Perceptrony wielowarstwowe są powszechnie stosowane w zadaniach takich jak klasyfikacja, regresja oraz rozpoznawanie wzorców. Ze względu na swoją elastyczność i moc obliczeniową, MLP stanowi fundament dla bardziej zaawansowanych struktur sieci neuronowych, takich jak sieci konwolucyjne i rekurencyjne. Jego zdolność do uczenia się nieliniowych relacji sprawia, że jest to narzędzie niezwykle użyteczne w szerokim zakresie zastosowań, od rozpoznawania obrazów po przetwarzanie języka naturalnego.

3.3.2. Sieci głębokie

Sieci głębokie (ang. Deep Neural Networks, DNN) to zaawansowane struktury sztucznych sieci neuronowych, które charakteryzują się dużą liczbą warstw ukrytych pomiędzy warstwą wejściową a wyjściową. Dzięki wielowarstwowej architekturze, sieci głębokie mogą modelować bardzo złożone i nieliniowe relacje w danych. Proces uczenia tych sieci wykorzystuje zaawansowane techniki, takie jak wsteczna propagacja błędów oraz optymalizatory gradientowe, które umożliwiają skuteczne dostosowanie wag neuronów. Sieci głębokie są niezwykle efektywne w zadaniach związanych z przetwarzaniem dużych ilości danych, takich jak rozpoznawanie obrazów, analiza dźwięku czy przetwarzanie języka naturalnego. Wprowadzenie warstw konwolucyjnych i rekurencyjnych w ramach sieci głębokich dodatkowo rozszerza ich możliwości, umożliwiając analizę sekwencji i wykrywanie istotnych wzorców w danych przestrzennych. Rozwój technologii sprzętowych, takich jak GPU, oraz technik takich jak dropout, znacząco przyczynił się do sukcesu i szerokiego zastosowania sieci głębokich. Dzięki swojej zdolności do automatycznego ekstrakowania cech i wysokiej dokładności predykcji, sieci głębokie stanowią fundament współczesnej sztucznej inteligencji i uczenia maszynowego.

3.3.3. Funkcja aktywacji softmax

Funkcja aktywacji softmax jest powszechnie stosowana w sieciach neuronowych, szczególnie w warstwach wyjściowych modeli klasyfikacyjnych, aby przekształcić surowe wartości wyjściowe neuronów w prawdopodobieństwa. Softmax przekształca wektor wartości rzeczywistych w wektor wartości, które sumują się do 1, co umożliwia interpretację wyniku jako rozkład prawdopodobieństwa między różnymi klasami. Wartości wyjściowe są eksponowane i następnie normalizowane, co podkreśla różnice między nimi i pomaga w dokładniejszej klasyfikacji. Dzięki swoim właściwościom, funkcja softmax jest idealna do zadań wieloklasowej klasyfikacji, gdzie model musi przypisać jedno z wielu możliwych oznaczeń do danego wejścia.

4. Techniki SHAP i LIME

Zagadnienia uczenia maszynowego postrzegane są niekiedy jako tzw. czarne skrzynki. Przekonanie to wzięło się z tego że, o ile prostsze modele są łatwo interpretowalne dla człowieka, o tyle te bardziej skomplikowane nie dają już żadnych wskazówek jakie cechy okazały się być decydujące jeśli chodzi o podjęte przez wybrany algorytm decyzje. Z pomocą przychodzą nam techniki SHAP i LIME. Pozwalają one na przybliżoną ocenę które z cech miały wpływ na wynik klasyfikacji.

4.1. SHAP

4.2. LIME (Local Interpretable Model-agnostic Explanations)

Lokalne modele zastępcze to modele interpretowalne, które służą do wyjaśniania pojedynczych przewidywań modeli uczenia maszynowego działających jak "czarne skrzynki". Praca dotycząca lokalnych, agnostycznych względem modelu wyjaśnień interpretowalnych (LIME) proponuje konkretną implementację lokalnych modeli zastępczych. Modele zastępcze są trenowane w celu przybliżenia przewidywań leżącego u podstaw modelu "czarnej skrzynki". Zamiast trenować globalny model zastępczy, LIME skupia się na szkoleniu lokalnych modeli zastępczych, aby wyjaśnić indywidualne przewidywania.

Podejście jest dość intuicyjne. Najpierw zapominamy o danych treningowych i wyobrażamy sobie, że mamy tylko model "czarnej skrzynki", do którego możesz wprowadzać dane i otrzymywać przewidywania modelu. Możemy testować skrzynkę tak często, jak chcemy. Naszym celem jest zrozumienie, dlaczego model uczenia maszynowego dokonał pewnego przewidywania. LIME testuje, co dzieje się z przewidywaniami, gdy wprowadzasz do modelu uczenia maszynowego warianty swoich danych. LIME generuje nowy zestaw danych składający się z zakłóconych próbek i odpowiadających im przewidywań modelu "czarnej skrzynki". Na tym nowym zestawie danych a następnie trenuje model interpretowalny, który jest ważony przez bliskość próbek do analizowanego przypadku. Modelem interpretowalnym może być dowolny model z rozdziału o modelach interpretowalnych, na przykład Lasso lub drzewo decyzyjne. Nauczony model powinien być dobrą lokalną aproksymacją przewidywań modelu uczenia maszynowego, ale nie musi być dobrą aproksymacją globalną. Tego rodzaju dokładność nazywa się także lokalną wiernością.

Matematycznie, lokalne modele zastępcze z ograniczeniem interpretowalności można wyrazić następująco:

$$explanation(x) = \arg \min_{g \in G} L(f, g, \pi_x) + \Omega(g) \quad (9)$$

Model wyjaśniający dla instancji x to model g (np. model regresji liniowej), który minimalizuje stratę L (np. błąd średniokwadratowy), mierząc jak

blisko wyjaśnienie jest do przewidywania oryginalnego modelu f (np. model `xgboost`), przy jednoczesnym zachowaniu niskiej złożoności modelu $\Omega(g)$ (np. preferowanie mniejszej liczby cech). G to rodzina możliwych wyjaśnień, na przykład wszystkie możliwe modele regresji liniowej. Miara bliskości π_x definiuje, jak duże sąsiedztwo wokół instancji x rozważamy dla wyjaśnienia. W praktyce LIME optymalizuje tylko część straty. Użytkownik musi określić złożoność, np. wybierając maksymalną liczbę cech, które może użyć model regresji liniowej.

LIME może być stosowany zarówno dla danych tabelarycznych, obrazów oraz tekstu.

4.2.1. LIME dla danych tabelarycznych i klasyfikacji wieloklasowej

Dane tabelaryczne to dane przedstawione w tabelach, gdzie każdy wiersz reprezentuje jedną instancję, a każda kolumna - cechę. Próbkę LIME nie są pobierane wokół interesującej instancji, lecz z centralnej masy danych treningowych, co jest problematyczne. Jednakże zwiększa to prawdopodobieństwo, że wynik dla niektórych punktów próbki będzie różnił się od punktu danych, który nas interesuje (instancji), i że LIME będzie w stanie przynajmniej w pewnym stopniu wyjaśnić wynik klasyfikacji.

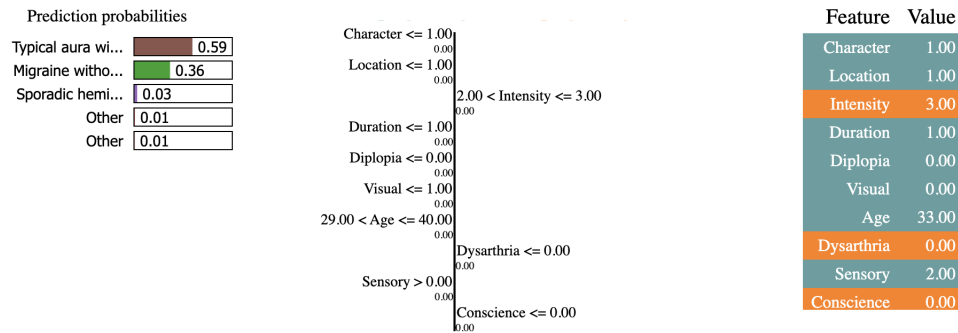
Dane użyte do wyjaśnienia techniką LIME nie powinny być normalizowane czy skalowane. Stanowi to pewnego rodzaju problem gdyż musimy budować nowy model (np. dla naiwnego klasyfikatora Bayesa) aby uniknąć pomyłki z tym bardziej optymalnym już znormalizowanym.

Po definicji explainera (w wolnym tłumaczeniu "wyjaśniacza"):

```
1 #Stworz explainer z danych treningowych (potrzebuje surowych,  
2                                     nieprzeskalowanych danych)  
3 explainer = LimeTabularExplainer(  
4     training\_data=X\_train.values, # nieprzeskalowane dane treningowe  
5     feature_names=df.columns[:-1].tolist(), # nazwy cech  
6     class\_names=class\_labels, # nazwy klas  
7     mode='classification' # dla zadania klasyfikacji  
8 )
```

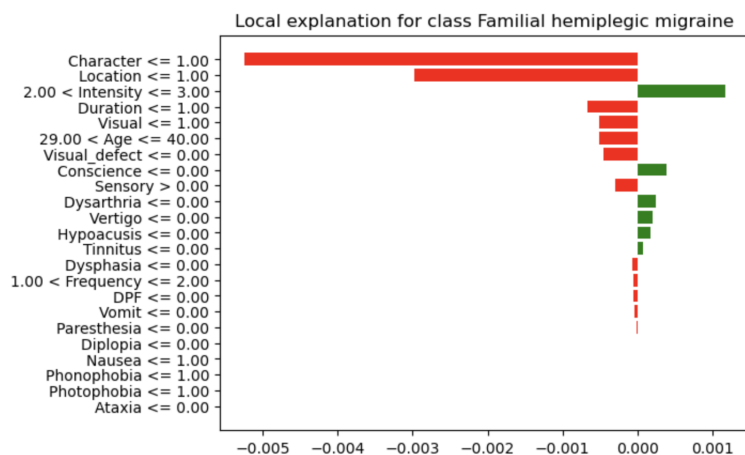
Listing 1. Definicja explainera

Następnie wybieramy instancję do interpretacji (najczęściej definiujemy iterator wskazujący na numer instancji w zbiorze testowym) i otrzymujemy graficzne wyjaśnienie podjętej przez algorytm decyzji:



Rysunek 2. Graficzna interpretacja decyzji modelu

Po stronie lewej widzimy tabelę podobieństwa do konkretnych klas. Po prawej wartości cech uszeregowane od najistotniejszych wg techniki LIME. Najwidzimy np. Po wartości wiersza "Age" dane nie są skalowane ani normalizowane. Na środku jest najistotniejszy element interpretacji, który akurat w naszym wypadku jest wyjątkowo nieczytelny ze względu na ograniczone możliwości skalowania osi. Natomiast istnieje jeszcze inna możliwość wyświetlenia tego wykresu:



Rysunek 3. Szczegółowy wykres istotności cech

Na osi pionowej widzimy nazwy cech wraz z przedziałami ich wartości dla analizowanej instancji. Na osi poziomej widzimy orientacyjną wartość na cecha ta wpływała na wynik - zarówno jeśli cecha świadczyła za przynależnością do klasy (wartości dodatnie oznaczone na zielono) jak i wpływ cech przeciw klasyfikacji (ujemne wartości oznaczone na czerwono). Dla czytelności wybrałem tabelę z innego przykładu. Widzimy jak pomiędzy obrazkiem nr 3 pokrywają się cechy z lewej i prawej strony osi wobec tego widzimy skróconej interpretacji z obrazka nr 2.

5. Eksperymenty

5.1. Klasyfikacja kNN przy podziale zbioru 80:20

W przeprowadzonym eksperymencie dokonano podziału zbioru 80:20 (20 procent danych testowych). Następnie zdefiniowano klasyfikator z parametrem `n_neighbors` równym 5. Kolejnym krokiem było wytrenowanie klasyfikatora i przeprowadzenie predykcji na zbiorze testowym:

```
1 from sklearn.neighbors import KNeighborsClassifier
2 from sklearn import metrics
3
4 knn = KNeighborsClassifier(n_neighbors=5)
5 knn.fit(X_train, y_train)
6
7 y_pred = knn.predict(X_test)
```

Listing 2. Definicja i użycie kNN

5.1.1. Wyniki

Podstawowe metryki dla pierwszego eksperymentu wynosiły precyzja (*Precision*): 0.85 a dokładność (*Accuracy*): 0.7875. Co nie stanowi wiele informacji, korzystam więc z gotowych metryk dostępnych w bibliotece scikit-learn:

	precision	recall	f1-score	support
Basilar-type aura	1.00	0.29	0.44	7
Familial hemiplegic migraine	0.50	0.80	0.62	5
Migraine without aura	0.62	0.62	0.62	8
Other	1.00	0.67	0.80	3
Sporadic hemiplegic migraine	0.00	0.00	0.00	4
Typical aura with migraine	0.82	0.94	0.88	49
Typical aura without migraine	1.00	1.00	1.00	4
accuracy			0.79	80
macro avg	0.71	0.62	0.62	80
weighted avg	0.77	0.79	0.76	80

Jak widzimy na powyższym listingu (i o czym również informuje nas warning biblioteki scikit-learn) klasa Sporadic hemiplegic migraine nie wystąpiła w zbiorze testowym ani razu. Zbiór nie jest ani specjalnie liczny ani też zbalansowany więc sytuacja taka jest jest zaskoczeniem ale nie przestaje to podważać efektywności walidacji naszych badań. W kolejnym eksperymencie przedstawiam za to inne proporcje podziały zbioru.

5.2. Klasyfikacja kNN przy podziale zbioru 75:25

Przebieg eksperymentu był dokładnie taki sam jak w poprzednim z tym że podział zbioru na dane uczące i testowe dokonał się w proporcjach 75:25 (tak

jak poprzednio ze sztuczną losowością aby zapewnić powtarzalność wyników).

5.2.1. Wyniki

Zmiana podziału zbioru nie wpłynęła zasadniczo na wartość precyzji i dokładności ale nie istniał już problem braku klas w zbiorze testowym, co nie przekreśla już sensowności przeprowadzonych obliczeń. Nadal jednak jeden z obiektów klasy "Sporadic hemiplegic migraine" nie został poprawnie rozpoznany co dalej, dla tego podzbioru klas zarówno precyzje jak i recall równe zero.

	precision	recall	f1-score	support
Basilar-type aura	1.00	0.12	0.22	8
Familial hemiplegic migraine	0.57	0.80	0.67	5
Migraine without aura	0.64	0.64	0.64	11
Other	1.00	0.50	0.67	4
Sporadic hemiplegic migraine	0.00	0.00	0.00	5
Typical aura with migraine	0.77	0.94	0.85	62
Typical aura without migraine	1.00	0.60	0.75	5
accuracy			0.75	100
macro avg	0.71	0.51	0.54	100
weighted avg	0.75	0.75	0.71	100

Ogólna wartość precyzji (*Precision*) dla całego zbioru testowego pozostała na poziomie 85%. Dokładność (*Accuracy*) na zbioru testowego spadła o 3,75 punktu procentowego (do poziomu 75%).

W przypadku walidacji krzyżowej precyzja wahała się pomiędzy 57 a 100% (w stosunku do poprzedniego podziału nieznaczne wzrosty w przypadku mniej licznych klas). Recall charakteryzował się dużym rozrzutem wartości - mniej liczne klasy miały recall niekiedy i 0-12% a te bardziej liczne 80-90%.

Średnia wartość recallu to 51% a średnia wazona 75% (ta właśnie uwzględniała nierówną liczebność klas). Podobna różnica miała miejsce w przypadku precyzji wyliczonej przy zastosowaniu walidacji krzyżowej - średnia wartość 71% - średnia wazona 75%.

5.3. Klasyfikacja Naiwnym Klasyfikatorem Bayesa

W eksperymencie tym sprawdziłem wyniki klasyfikacji Naiwnym Klasyfikatorem Bayesa. Biblioteka scikit-learn umożliwia przeprowadzenie tzn *Categorical Classification (Categorical Naive Bayes)* gdzie w przeciwieństwie do popularnej klasyfikacji binarnej, metoda jest w stanie zwrócić dwie informacje: etykietę klasy będącej wynikiem klasyfikacji oraz listę ze stopniami

podobieństwa dla wszystkich klas.

```
1 from sklearn.naive_bayes import CategoricalNB
2
3 gnb = CategoricalNB()
4
5 y_pred = gnb.fit(X_train, y_train).predict(X_test)
```

Listing 3. Definicja i użycie CategoricalNB

5.3.1. Wyniki

O ile wyniki dokładności (*Accuracy*) wyglądały podobnie niz dla np. kNN - dokładnie 75%. O tyle wyniki dla całości zbioru prezentowały się gorzej o 5-6 punktów procentowych niz dla kNN - precyzja średnia 64% (względem 71%) oraz precyzja wazona 69% (względem 75%). Wyniki dla poszczególnych klas dla precyzji potrafiły się różnić o 20-30 punktów procentowych względem kNN nie dając wyraźniej korelacji względem liczebności klas. Wyniki dla recall po trafiły być niekiedy diametralnie różne - 88%-12% a niekiedy znikomo - 92%-94% względem kNN.

	precision	recall	f1-score	support
Basilar-type aura	1.00	0.88	0.93	8
Familial hemiplegic migraine	0.75	0.60	0.67	5
Migraine without aura	0.29	0.18	0.22	11
Other	1.00	1.00	1.00	4
Sporadic hemiplegic migraine	0.00	0.00	0.00	5
Typical aura with migraine	0.76	0.92	0.83	62
Typical aura without migraine	0.67	0.40	0.50	5
accuracy			0.75	100
macro avg	0.64	0.57	0.59	100
weighted avg	0.69	0.75	0.71	100

Niezwykle istotną możliwością wykorzystania Naiwnego Klasyfikatora Bayesa było wyświetlenie podobieństwa testowanej klasy względem wszystkich klas.

Niektóre z klas (niekoniecznie te bardziej licznie) przedstawiały bardzo jednoznaczną klasyfikację, np:

```
Other
0.0009411289994916489: Basilar-type aura
0.00024106562936048384: Familial hemiplegic migraine
1.3757510137960067e-06: Migraine without aura
0.9987800638900048: Other
5.212399025390643e-06: Sporadic hemiplegic migraine
3.406207926665455e-08: Typical aura with migraine
```

3.111926902521578e-05: Typical aura without migraine

dla innych (często dla konkretnych jak np. Typical aura without migraine czy Migraine without aura) mieliśmy rozkłady bardzo zróżnicowane:

Typical aura without migraine

0.08048228203931468: Basilar-type aura
0.2633221828713967: Familial hemiplegic migraine
0.0009416760319083586: Migraine without aura
0.005682255909626338: Other
0.22367861315019222: Sporadic hemiplegic migraine
0.15361555787258202: Typical aura with migraine
0.2722774321249797: Typical aura without migraine

albo nawet niemal bilateralne:

Migraine without aura

0.000315062484758524: Basilar-type aura
0.0008466488192796888: Familial hemiplegic migraine
0.516090617842986: Migraine without aura
0.001079067159295874: Other
0.0038615304383266213: Sporadic hemiplegic migraine
0.4772101808065339: Typical aura with migraine
0.0005968924488194133: Typical aura without migraine

5.4. Selekcja i ekstrakcja cech

W czasie wykonanych eksperymentów wykorzystano dwa sposoby selekcji cech: SKB i RFE, oraz mechanizm wyodrębnienia cech PCA.

Definicja algorytmu selekcji cech SKB (czyli Select k Best), wybór k dzieśięciu najlepszych i wyświetlenie nowych nazw cech:

```
1 from sklearn.feature_selection import SelectKBest, chi2
2
3 skb = SelectKBest(chi2, k = 10)
4 X2 = skb.fit_transform(X, y)
```

Listing 4. Definicja selektora SKB

Definicja selektora cech RFE (dla klasyfikatora SVC i redukcji do 10 cech):

```
1 from sklearn.feature_selection import RFE
2
3 svc = SVC(kernel = 'linear')
4
5 rfe = RFE(estimator = svc, n_features_to_select = 10, step = 1)
6 X3 = rfe.fit_transform(X, y.values.ravel())
```

Listing 5. Definicja selektora RFE

Definicja ekstraktora cech PCA (ekstrakcja do 4 cech).

```
1 from sklearn.decomposition import PCA
2
3 pca = PCA(n_components = 4)
```



```
4 X4 = pca.fit_transform(X)
```

Listing 6. Definicja ekstraktora cech PCA

5.4.1. Wyniki

Wyniki prezentowały się następująco:

Średnia dokładność klasyfikacji z wykorzystaniem klasyfikatora ...

kNN wyniosła:	0.8024999999999999
kNN i metody SKB wyniosła:	0.7825
kNN i metody RFE wyniosła:	0.865
kNN i metody PCA:	0.7
SVC wyniosła:	0.8825
SVC i metody SKB wyniosła:	0.8
SVC i metody RFE wyniosła:	0.8825
SVC i metody PCA:	0.735

Co interesujące to właśnie dyskusyjny eksperyment klasyfikacji KNN z selekcją RFE poprawił wynik dla kNN o 8 punktów procentowych a np. SKB pogorszył go o dwa. W przypadku klasyfikatora SVC w/w metody albo nie zmieniły wyniku (RFE) albo pogorszyły wynik od 8-10 punktów procentowych.

Użyty zbiór nie był zbiorem licznym ani też obliczenia na nim wykonywane nie wymagały dużych mocy obliczeniowych. Nie istniało zatem żadne uzasadnienie stosowania którejkolwiek z w/w metod ponieważ przy praktycznej zerowej optymalizacji otrzymywaliśmy gorsze wyniki.

5.5. Perceptron wielowarstwowy

Kolejnym z przeprowadzonych eksperymentów było użycie perceptronu wielowarstwowego. Podział zbioru to 75% na dane treningowe i 25% na dane testowe. Ponownie wykorzystaliśmy bibliotekę scikit-learn:

```
1 from sklearn.neural_network import MLPClassifier
2 mlp = MLPClassifier(hidden_layer_sizes=(6),
3                     max_iter=50000,
4                     alpha=0.0001,
5                     solver='adam',
6                     activation='logistic',
7                     random_state=25,
8                     tol=0.0000001)
```

Listing 7. Definicja perceptronu wielowarstwowego

Wyjaśnienie wykorzystanych parametrów:

1. hidden_layer_sizes - ilość warstw ukrytych
2. max_iter - liczba epok
3. alpha - współczynnik regularyzacji który zapobiega przeuczenia

4. solver - wybór algorytmu optymalizacji "adam".
5. activation - wybór sigmoidalnej funkcji aktywacji
6. random_state - ziarno generatora liczb losowych
7. tol - tolerancja dla kryterium stop. Algorytm zatrzymuje się, jeśli zmiana wyniku (kosztu) jest mniejsza niż ta wartość.

5.5.1. Wyniki

Wyniki działania perceptronu okazały się zaskakująco dobre. Przy relatywnie niskiej ilości warstw ukrytych (6), przeciętnej ilości epok (50000) i popularnym optymalizatorze ("adam") osiągnęliśmy dokładność (Accuracy) na poziomie 94% i średnią ważoną precyzję 95%. Recall wahał się od 60% do 100% (przy średniej ważonej wartości 94%) a współczynnik F1 od 75% do 100% (przy średniej ważonej wartości 93%).

	precision	recall	f1-score	support
Basilar-type aura	1.00	0.20	0.33	5
Familial hemiplegic migraine	0.57	0.80	0.67	5
Migraine without aura	1.00	1.00	1.00	12
Other	1.00	1.00	1.00	3
Sporadic hemiplegic migraine	0.75	1.00	0.86	3
Typical aura with migraine	0.97	0.99	0.98	67
Typical aura without migraine	1.00	1.00	1.00	5
accuracy			0.94	100
macro avg	0.90	0.86	0.83	100
weighted avg	0.95	0.94	0.93	100

NN Accuracy = 0.94

Przy mniejszej ilości epok (5000) takiej samej ilości warstw ukrytych (6) i tym samym optymalizatorze ("adam") osiągnęliśmy dokładność (Accuracy) na poziomie 96% i średnią ważoną precyzję 97%. Recall wahał się od 60% do 100% a współczynnik F1 od 75% do 100% (przy średniej ważonej wartości 96%) z tym że biblioteka wyświetliła ostrzeżenie że optymalizator nie zakończył działania więc wyniki mogą być nieoptymalne.

	precision	recall	f1-score	support
Basilar-type aura	1.00	0.60	0.75	5
Familial hemiplegic migraine	0.67	0.80	0.73	5
Migraine without aura	1.00	1.00	1.00	12
Other	1.00	1.00	1.00	3
Sporadic hemiplegic migraine	0.75	1.00	0.86	3
Typical aura with migraine	0.99	0.99	0.99	67
Typical aura without migraine	1.00	1.00	1.00	5

accuracy			0.96	100
macro avg	0.91	0.91	0.90	100
weighted avg	0.97	0.96	0.96	100

NN Accuracy = 0.96

5.6. Sieć głęboka z funkcją aktywacji Softmax

Perceptron wielowarstwowy choć dawał świetne wyniki to jednak dawał wynik przynależności do najbardziej prawdopodobnej klasy. W zagadnieniu klasyfikacji bólów głowy interesującym byłoby pokazać wynik w postaci stopnia podobieństwa przynależności do każdej z możliwych klas. Z pomocą przychodzi nam funkcja aktywacji Softmax i możliwość zastosowania jej w głębokich sieciach neuronowych.

W przypadku funkcji aktywacji Softmax konieczne jest użycie enkodera etykiet klas - czy zamianę etykiet na przyporządkowane im liczby. Następnie w/w liczby zamieniamy na tzw. format hot-one czyli format gdzie danej liczbie przyporządkowujemy ciąg binarny zer oraz jednej jedynki w miejscu w szeregu odpowiadającym danej liczbie. Po testowej klasyfikacji musimy nasze wyniki zdekodować z powrotem do formatu etykiet. Wszystkie te operacje umożliwia nam biblioteka scikit-learn.

Do definicji sieci głębokiej używam biblioteki TensorFlow i jej części Keras.

```

1 model=tf.keras.models.Sequential()
2 model.add(tf.keras.layers.Dense(64, activation='relu'))
3 model.add(tf.keras.layers.Dense(32, activation='relu'))
4 model.add(tf.keras.layers.Dense(num_classes,
5                                 activation='softmax'))
6 model.build(input_shape=23)
7 model.compile(optimizer='Adam',
8               loss='categorical_crossentropy',
9               metrics=['accuracy'])

```

Listing 8. Definicja sieci głębokiej z funkcją aktywacji Softmax

W powyższej definicji mamy:

1. definicje modelu sekwencyjnego
2. definicje warstwy gęstej (Dense, wszystkie neurony połączone) z 64 neuronami i funkcją aktywacji ReLU.
3. definicje warstwy gęstej (Dense, wszystkie neurony połączone) z 32 neuronami i funkcją aktywacji ReLU.
4. definicje warstwy gęstej (Dense) z funkcją aktywacji Softmax.
5. określenie ilości wejść - 23 bo tyle mamy cech.
6. kompilację modelu z optymalizatorem "adam", określeniem funkcji straty "categorical_crossentropy" i definicje metryk.

Następnie rozpoczynam trenowanie modelu dla 50 epok, wydzielenie 20% zbioru testowego na zbiór walidacyjny i zapis modelu:

```

1 history = model.fit(X_train,

```

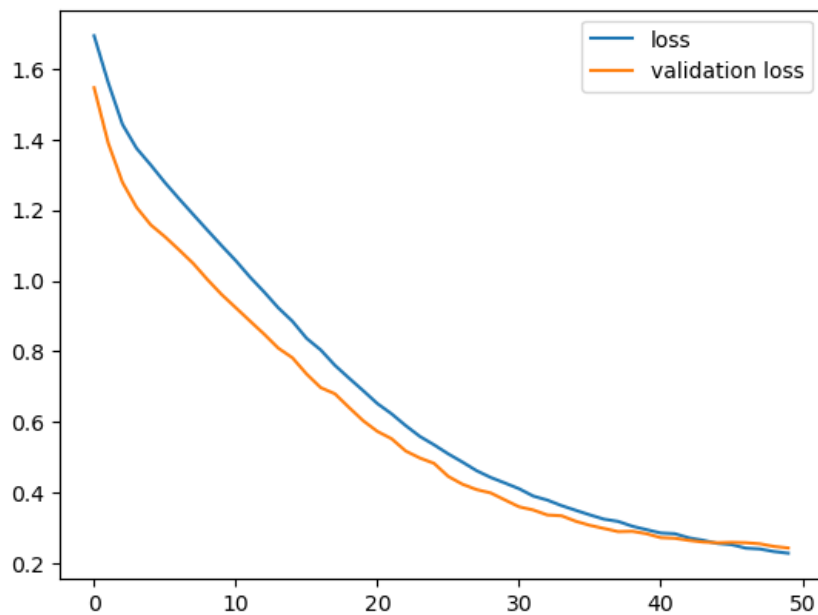
```

2         y_train_one_hot,
3         epochs=50,
4         verbose=1,
5         validation_split=0.2)
6 tf.keras.models.save_model(model, '../models/softmax.keras')

```

Listing 9. Definicja sieci głębokiej z funkcją aktywacji Softmax

Wyświetlenie wykresu wartości funkcji straty i porównanie jej z wartością straty na zbioru walidacyjnego daje mi orientacyjną możliwość oceny słuszności doboru ilości epok.



Rysunek 4. Wykresy funkcji strat

5.6.1. Wyniki

Dla zbioru testowego osiągnęliśmy 95% dokładność.

```

4/4 ===== 0s 801us/step - accuracy: 0.9446 - loss: 0.2267
Loss: 0.21024955809116364, Accuracy: 0.949999988079071

```

Najciekawszym aspektem wykorzystania funkcji Softmax jest możliwość wyświetlenia podobieństwa do wszystkich możliwych klas. Oto kilka przykładów dla konkretnych przypadków, co zrobiłem na dwa sposoby. Pierwszy z nich wyświetlał wszystkie klasy i podobieństwa konkretnego przypadków do każdej z nich:

Class: Basilar-type aura,	Probability: 0.25
Class: Familial hemiplegic migraine,	Probability: 0.63
Class: Migraine without aura,	Probability: 0.00
Class: Other,	Probability: 0.09

Class: Sporadic hemiplegic migraine, Probability: 0.03
 Class: Typical aura with migraine, Probability: 0.00
 Class: Typical aura without migraine, Probability: 0.00

a drugi z nich wyświetlał tabelę: enkodowana etykieta klasy, pełna etykieta klasy i prawdopodobieństwo do danej klasy analizowanego przypadku. Wartości uszeregowane malejąco wg podobieństwa:

	Type	Probabilities
1	Familial hemiplegic migraine	0.63
0	Basilar-type aura	0.25
3	Other	0.09
4	Sporadic hemiplegic migraine	0.03
5	Typical aura with migraine	0.00
6	Typical aura without migraine	0.00
2	Migraine without aura	0.00

Spis rysunków

1. Schemat działania SVM	7
2. Graficzna interpretacja decyzji modelu	12
3. Szczegółowy wykres istotności cech	12
4. Wykresy funkcji strat	20

Literatura

- [1] https://www.researchgate.net/publication/291331282_The_International_Classification_of_Headache_Disorders_3rd_edition_beta_version
- [2] <https://www.ehf-headache.com/>
- [3] <https://link.springer.com/article/10.1186/s10194-018-0909-4>
- [4] <https://codeocean.com/capsule/1269964/tree/v1>
- [5] <https://docs.python.org/3/license.html>
- [6] <https://github.com/pandas-dev/pandas/blob/main/LICENSE>
- [7] <https://github.com/numpy/numpy/blob/main/LICENSE.txt>
- [8] <https://github.com/mwaskom/seaborn/blob/master/LICENSE.md>
- [9] <https://github.com/tensorflow/tensorflow/blob/master/LICENSE>
- [10] Van Rossum, Guido and Drake, Fred L.
"Python 3 Reference Manual", 2009
ISBN 1441412697
- [11] <https://zenodo.org/records/10957263>
- [12] <https://doi.org/10.1038/s41586-020-2649-2>
- [13] <https://joss.theoj.org/papers/10.21105/joss.03021>
- [14] <https://zenodo.org/records/10798587>
- [15] Materiały wykładowe przedmiotu.
- [16] https://pl.wikipedia.org/wiki/Maszyna_wektor%C3%B3w_no%C5%9Bnych
- [17] <https://christophm.github.io/interpretable-ml-book/lime.html>