Programowanie sieciowe

Klasyfikacja wieloklasowa MLP - Lab2

Data:	8.05.2022	Dzień:	Wtorek TN + 1/2
Grupa:	Y02-15b	Godzina:	17:05
Numer indeksu:	252889	Prowadzący:	dr inż. Marek Bazan
Nazwisko i imię: Nowek Jakub			

Spis treści

1. Opis problemu

Należało dokonać klasyfikacji wieloklasowej na zbiorach danych przy użyciu perceptronu wielowarstwowego z jedną warstwą ukrytą. W zadaniu należało skorzystać z pakietu scikit-learn języka Python.

1.1. Zad.1

W pierwszym zadaniu należało zaimplementować klasyfikację trzech klas dla danych Iris.

1.2. Zad.2

W drugim zadaniu należało zaimplementować klasyfikację dziesięciu klas dla zestawu danych MNIST, przedstawiającego cyfry od 0 do 9. Zbiór danych MNIST składa się z 70 000 czarno-białych zdjęć przedstawiających cyfry od 0 do 9 pisane pismem odręcznym.

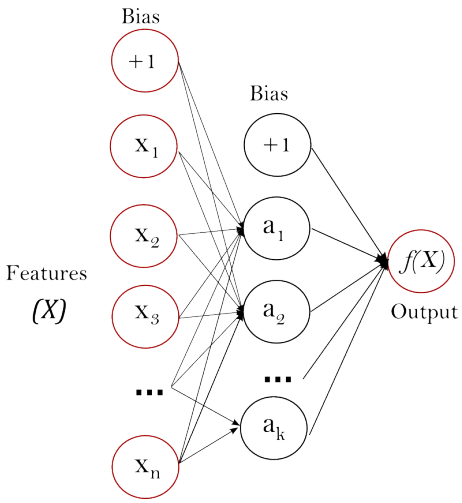
1.3. Zad.3

W trzecim zadaniu zaimplementowano klasyfikację trzech klas obiektów gwiezdnych: gwiazd, galaktyk i kwazarów. Pod uwagę wzięto 30 300 różnych, losowo wybranych ze zbioru 100 000 obiektów z bazy Stellar Classification Dataset - SDSS17 (`https://www.kaggle.com/fedesoriano/stellar-classification-dataset-sdss17`). W uczeniu rozważano 8 parametrów:

- alpha = Rektascensja - położenie ciała niebieskiego na sferze niebieskiej w układzie współrzędnych astronomicznych (epoka J2000)
- delta = Deklinacja - pomiędzy kierunkiem poprowadzonym od obserwatora do obiektu a płaszczyzną równika niebieskiego (epoka J2000)
- u = wartość filtru ultrafioletu w systemie fotometrycznym
- g = wartość filtru światła zielonego w systemie fotometrycznym
- r = wartość filtru światła czerwonego w systemie fotometrycznym
- i = wartość filtru promieniowania bliskiego podczerwieni w systemie fotometrycznym
- z = wartość filtru podczerwieni w systemie fotometrycznym
- redshift = przesunięcie ku czerwieni (inie widmowe promieniowania elektromagnetycznego docierające z niektórych gwiazd lub galaktyk są przesunięte w stronę większych długości fali)

2. Opis użytych algorytmów

Do stworzenia modelu perceptronu wykorzystać należało funkcję MLPClassifier. Funkcja ta do uczenia wykorzystuje **propagację wsteczną** - algorytm uczenia wyznacza kierunek, w którym w danej iteracji należy zmodyfikować wagi w celu zmniejszenia błędu popełnianego przez sieć. Tempo modyfikacji wag określone jest natomiast za pomocą współczynnika uczenia. Na poniższym rysunku znajduje się model tak rozumianego MLP z jednym wyjściem. W przypadku tego zadania użyto analogicznego modelu MLP o wielu wyjściach.



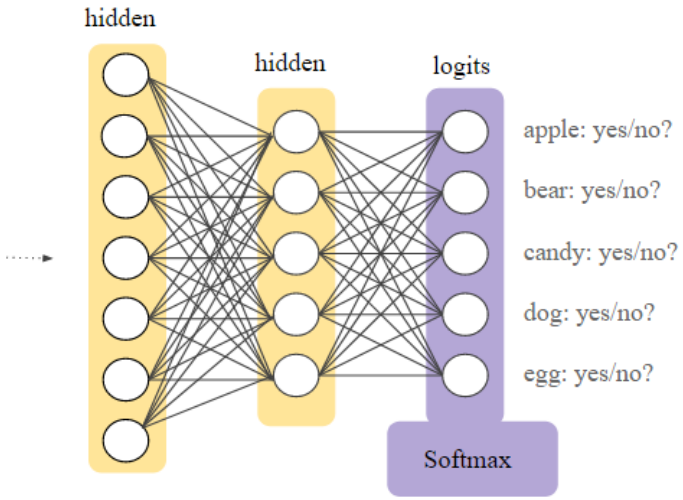
Rys. 1: Wielowarstwowy perceptron z jedną warstwą ukrytą i jednym wyjściem.

Funkcją aktywacji w tym przypadku był tangens hiperboliczny (tanh(x)). Jako wyjściowej funkcji aktywacji użyto funkcji softmax. Softmax przypisuje każdej klasie na wyjściu, wartości prawdopodobieństwa wystąpienia danej klasy. Prawdopodobieństwa wszystkich klas dla jednej próbki muszą się sumować do wartości "1". Dodatkowo kodowanie wyjść wykorzystuje "gorącą jedynkę" (ang. 'hot one'), która wyjściu z największym prawdopodobieństwem przypisuje wartość 1, a na pozostałych wartość 0.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ dla } i = 1, \ldots, K \text{ i } z = (z_1, \ldots, z_K) \in R^K$$

Rys. 2: Wzór funkcji softmax.

Poniżej przedstawiono model wyjściowej funkcji aktywacji softmax.



Rys. 3: Model z funkcją softmax.

(UWAGA) Na powyższym modelu MLP ma więcej niż jedną warstwę ukrytą, w naszym przypadku jest tylko jedna warstwa ukryta.

Do wyszukania najlepszych parametrów do użycia w funkcji MLPClassifier, została użyta funkcja GridSearchCV.

Funkcja GridSearchCV wyszukuje optymalnych parametrów dla modelu, spośród podanych. Zwraca ona listę modeli w których każdy ma inne parametry. Liczba modeli zwróconych przez funkcję będzie zatem wynikiem iloczynu ilości różnych wartości danego parametru. Trenowanie jest więc wykonywane przy pomocy podanych zbiorów dla każdego wyznaczonego przez funkcję modelu. Po zakończeniu treningu możemy odczytać wyniki testów przeprowadzonych przez funkcję GridSearch (metoda cv_results_).

Najbardziej interesującym parametrem zwracanym przez cv_results_ jest **mean_ test_score**. Jest to kolumna zawierająca medianę wyznaczonych dokładności dla każdej próbki, dla każdego modelu.

3. Testy numeryczne

3.1. Definicja testów

Dla każdego zestawu danych, trenowano algorytm za pomocą zbioru treningowego (wartości wejść i wyjść), oraz zbioru testowego, który zawierał dane inne niż w zbiorze treningowym. Dane do zbiorów były losowo wybierane z zestawu danych.

Dla każdego zestawu danych sprawdzana była dokładność klasyfikacji przy następujących parametrach:

- Liczba neuronów (hidden layer sizes=(N,)) – N $\in$ {20, 40, . . . , 100},
- Współczynnik uczenia sieci (learning rate init = η) – η = {0.1, 0.01, 0.001},
- Algorytm optymalizacji wag (solver = name) – name $\in$ {adam, lfbs, sgd}.

Wynik przedstawiany był w postaci tabeli mean_ test_score oraz macierzy pomyłek dla najskuteczniejszego z modeli.

3.2. Zad.1 Dane Iris.

Zbiór treningowy zawierał po 80% danych z każdej kasy zbioru Iris (łącznie 120 próbek, po 40 z każdej klasy). Pozostałe 20% z każdej klasy umieszczono w zbiorze testowym (łącznie 30 próbek, po 10 z każdej klasy).

3.3. Zad.2 MNIST.

Każde zdjęcie ze zbioru MNIST zostało wczytane do klasyfikatora jako macierz wektorów, z których każdy posiadał 784 elementy (rozmiar jednego zdjęcia to 28 pikseli * 28 pikseli). Wejście zbioru treningowego było zatem macierzą 60 000 wektorów o długości 784, a wyjście zbioru treningowego było zbiorem 60 000 wartości reprezentujących klasę do której przynależało dane zdjęcie. Wartość ta była równa liczbie którą reprezentowało dane zdjęcie (10 klas od 0 do 9). Zbiór testowy składał się z 10 000 wektorów (wejście) oraz rzeczywistych wartości klas dla testowanych próbek.

3.4. Zad3. Stellar.

Zbiór treningowy zawierał 30 000 danych obiektu z każdej klasy zbioru (po 10 000 z każdej klasy). Zbiór testowy składał się z 300 obiektów (po 100 dla każdej klasy). Oznaczenia klas były następujące:

- "0" - galaktyka
- "1" - gwiazda
- "2" - kwazar

W zbiorze testowym znajdowało się kolejno 100 galaktyk, 100 gwiazd oraz 100 kwazarów.

4. Wyniki.

Wyniki testów zaprezentowano na poniższych rysunkach.

4.1. Dane Iris.

Dla każdego solwera pozostawiono domyślny parametr max_iter=200.

4.1.1. Wyniki funkcji GridSearch.

Grid search results:

	params	mean_test_score	rank_test_score
0	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.958333	9
1	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.966667	5
2	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.941667	15
3	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.958333	9
4	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.966667	5
5	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.958333	9
6	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.958333	9
7	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.975000	3
8	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.983333	1
9	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.958333	9
10	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.966667	5
11	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.983333	1
12	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.958333	9
13	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.966667	5
14	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.975000	3

Rys. 4: Skuteczność w zależności od parametrów dla solvera ADAM.

Grid search results:

	params	mean_test_score	rank_test_score
0	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.750000	12
1	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.958333	6
2	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.941667	10
3	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.775000	11
4	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.975000	1
5	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.958333	6
6	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.741667	13
7	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.975000	1
8	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.966667	5
9	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.725000	14
10	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.975000	1
11	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.958333	6
12	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.716667	15
13	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.975000	1
14	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.958333	6

Rys. 5: Skuteczność w zależności od parametrów dla solvera SGD.

```
Grid search results:
                                                                 params  mean_test_score  rank_test_score
0      {'hidden_layer_sizes': 20, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.933333               12
1     {'hidden_layer_sizes': 20, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.958333                2
2    {'hidden_layer_sizes': 20, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}         0.950000                4
3      {'hidden_layer_sizes': 40, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.925000               15
4     {'hidden_layer_sizes': 40, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.933333               14
5    {'hidden_layer_sizes': 40, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}         0.966667                1
6      {'hidden_layer_sizes': 60, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.941667                9
7     {'hidden_layer_sizes': 60, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.933333               12
8    {'hidden_layer_sizes': 60, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}         0.941667                9
9      {'hidden_layer_sizes': 80, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.950000                4
10    {'hidden_layer_sizes': 80, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.950000                4
11   {'hidden_layer_sizes': 80, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}         0.950000                4
12    {'hidden_layer_sizes': 100, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.958333                2
13   {'hidden_layer_sizes': 100, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.950000                4
14  {'hidden_layer_sizes': 100, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}         0.941667                9
```

Rys. 6: Skuteczność w zależności od parametrów dla solvera LBFGS.

```
Grid search results:
                                                                params  mean_test_score  rank_test_score
0     {'hidden_layer_sizes': 2, 'learning_rate_init': 0.1, 'solver': 'adam'}         0.650000               15
1    {'hidden_layer_sizes': 2, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.908333                9
2    {'hidden_layer_sizes': 2, 'learning_rate_init': 0.01, 'solver': 'adam'}         0.600000               16
3   {'hidden_layer_sizes': 2, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.700000               14
4     {'hidden_layer_sizes': 3, 'learning_rate_init': 0.1, 'solver': 'adam'}         0.900000               11
5    {'hidden_layer_sizes': 3, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.908333                9
6    {'hidden_layer_sizes': 3, 'learning_rate_init': 0.01, 'solver': 'adam'}         0.958333                3
7   {'hidden_layer_sizes': 3, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.966667                2
8     {'hidden_layer_sizes': 4, 'learning_rate_init': 0.1, 'solver': 'adam'}         0.958333                3
9    {'hidden_layer_sizes': 4, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.958333                3
10   {'hidden_layer_sizes': 4, 'learning_rate_init': 0.01, 'solver': 'adam'}         0.958333                3
11  {'hidden_layer_sizes': 4, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.958333                3
12    {'hidden_layer_sizes': 5, 'learning_rate_init': 0.1, 'solver': 'adam'}         0.841667               13
13   {'hidden_layer_sizes': 5, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.975000                1
14   {'hidden_layer_sizes': 5, 'learning_rate_init': 0.01, 'solver': 'adam'}         0.950000                8
15  {'hidden_layer_sizes': 5, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.900000               11
```

Rys. 7: Skuteczność w zależności od parametrów - Porównanie solwerów dla małych ilości neuronów warstwy ukrytej.

4.1.2. Wyniki predykcji dla najlepszych parametrów wybranych przez GridSearch

Parametry:

- ilość neuronów w warstwie ukrytej: 80
- prędkość uczenia: 0.001

```
Prediction output:
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2]
Confusion matrix:
 [[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```

Rys. 8: Wynik predykcji dla solwera ADAM.

Parametry:

- ilość neuronów w warstwie ukrytej: 100
- prędkość uczenia: 0.01

```
Prediction output:
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2]
Confusion matrix:
 [[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```

Rys. 9: Wynik predykcji dla solwera SGD.

Parametry:

- ilość neuronów w warstwie ukrytej: 40
- prędkość uczenia: 0.001

```
Prediction output:
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2]
Confusion matrix:
 [[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```

Rys. 10: Wynik predykcji dla solwera LBFGS.

Parametry:

- ilość neuronów w warstwie ukrytej: 5
- prędkość uczenia: 0.1
- solver: LBFGS

```
Prediction output:
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 1 2 1 2 2 2 2 2 2 2]
Confusion matrix:
 [[10  1  0]
 [ 0  9  1]
 [ 0  0  9]]
```

Rys. 11: Wynik predykcji dla solwera LBFGS przy małej ilości neuronów w warstwie ukrytej.

4.1.3. Różnice w prawdopodobieństwie solwerów gradientowyh i LBFGS

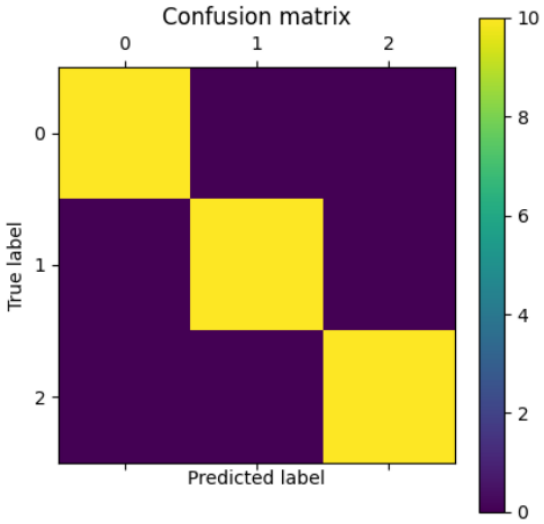
```
Prediction probability for test set:
 [[97.03405453  2.94433482  0.02161065]
 [89.48445901 10.38029347  0.13524753]
 [96.21402512  3.74504669  0.04092818]
 [93.10088448  6.8245848   0.07453073]
 [93.06289761  6.86700163  0.07010077]
 [94.12486568  5.81951182  0.0556225 ]
 [96.63675666  3.33884681  0.02439653]
 [95.57835684  4.37808578  0.04355739]
 [96.96583422  3.01573153  0.01843425]
 [96.16555449  3.80623361  0.02821191]
 [ 2.15022647 62.05155765 35.79821588]
 [ 2.41797568 64.93063107 32.65139325]
 [ 4.65366556 76.87245163 18.47388282]
 [ 9.44054527 78.19012722 12.36932751]
 [ 3.07446625 67.57279885 29.35273491]
 [ 4.73427546 73.21596126 22.04976328]]
```

(a) Prawdopodobieństwa dla solwera ADAM.

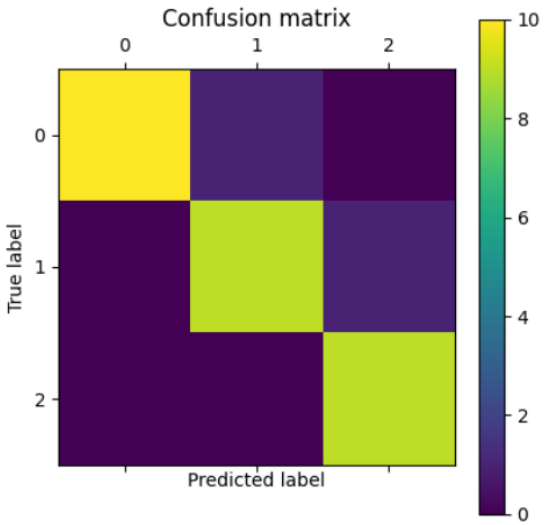
```
Prediction probability for test set:
 [[100.           0.           0.        ]
 [100.           0.           0.        ]
 [100.           0.           0.        ]
 [100.           0.           0.        ]
 [100.           0.           0.        ]
 [100.           0.           0.        ]
 [100.           0.           0.        ]
 [100.           0.           0.        ]
 [100.           0.           0.        ]
 [100.           0.           0.        ]
 [  0.         100.           0.        ]
 [  0.          99.99980306   0.00019694]
 [  0.         100.           0.        ]
 [  0.         100.           0.        ]
 [  0.          99.9999998    0.0000002 ]
 [  0.          99.99999291   0.00000709]
 [  0.          99.99998941   0.00001059]]
```

(b) Prawdopodobieństwa dla solwera LBFGS.

4.2. Porównanie macierzy pomyłek dla algorytmów o dużej i małej liczbie neuronów w warstwie ukrytej.



(c) Solwery o liczbie neuronów > 20.



(d) Solwery o liczbie neuronów < 20

4.3. Dane MNIST.

Dla każdego solwera ustawiono parametr max_iter=20.

4.3.1. Wyniki funkcji GridSearch.

Grid search results:

	params	mean_test_score	rank_test_score
0	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.315750	15
1	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.740517	10
2	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.882650	5
3	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.442800	14
4	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.774000	9
5	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.903617	4
6	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.445233	13
7	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.785317	8
8	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.917967	3
9	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.487600	12
10	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.817867	6
11	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.918383	2
12	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.525267	11
13	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.804700	7
14	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.922083	1

Rys. 12: Skuteczność w zależności od parametrów dla solvera ADAM.

Grid search results:

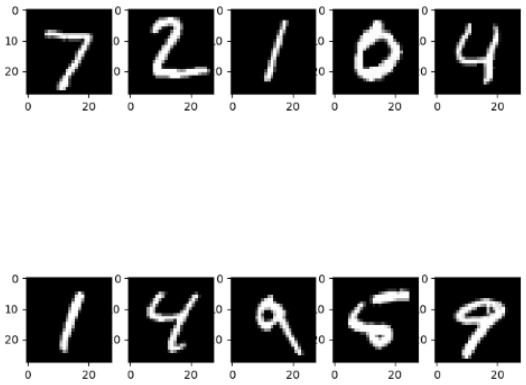
	params	mean_test_score	rank_test_score
0	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.239833	15
1	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.729200	10
2	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.876717	5
3	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.254517	14
4	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.804033	9
5	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.909500	4
6	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.413983	11
7	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.812883	8
8	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.918000	3
9	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.395450	12
10	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.837683	7
11	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.925267	2
12	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.349083	13
13	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.857167	6
14	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.929850	1

Rys. 13: Skuteczność w zależności od parametrów dla solvera SGD.

```
Grid search results:
                                                                                params  mean_test_score  rank_test_score
0      {'hidden_layer_sizes': 20, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.832817               13
1     {'hidden_layer_sizes': 20, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.821167               15
2    {'hidden_layer_sizes': 20, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}         0.823233               14
3      {'hidden_layer_sizes': 40, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.868517               12
4     {'hidden_layer_sizes': 40, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.870083               11
5    {'hidden_layer_sizes': 40, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}         0.870283               10
6      {'hidden_layer_sizes': 60, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.882383                8
7     {'hidden_layer_sizes': 60, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.883650                7
8    {'hidden_layer_sizes': 60, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}         0.881033                9
9      {'hidden_layer_sizes': 80, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.893433                4
10    {'hidden_layer_sizes': 80, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.893133                6
11   {'hidden_layer_sizes': 80, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}         0.893250                5
12    {'hidden_layer_sizes': 100, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}         0.896633                2
13   {'hidden_layer_sizes': 100, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}         0.896117                3
14  {'hidden_layer_sizes': 100, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}         0.899833                1
```

Rys. 14: Skuteczność w zależności od parametrów dla solvera LBFGS.

4.3.2. Wyniki predykcji dla najlepszych parametrów wybranych przez GridSearch



Rys. 15: Rzeczywiste wartości danych testowych.

Parametry:

- ilość neuronów w warstwie ukrytej: 100
- prędkość uczenia: 0.001

```
Prediction output:
 [7 2 1 0 4 1 4 9 6 9]
Confusion matrix:
 [[1 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0]
 [0 0 0 2 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 2 0]]
```

Rys. 16: Wynik predykcji dla solwera ADAM.

Parametry:

- ilość neuronów w warstwie ukrytej: 100
- prędkość uczenia: 0.001

```
Prediction output:
 [7 2 1 0 4 1 4 9 6 9]
Confusion matrix:
 [[1 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0]
 [0 0 0 2 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 2 0]]
```

Rys. 17: Wynik predykcji dla solwera SGD.

Parametry:

- ilość neuronów w warstwie ukrytej: 100
- prędkość uczenia: 0.001

```
Prediction output:
 [7 2 1 0 4 1 4 9 6 9]
Confusion matrix:
 [[1 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0]
 [0 0 0 2 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 2 0]]
```

Rys. 18: Wynik predykcji dla solwera LBFGS.

Predykcje dla solwera ADAM, przy parametrze max_iters równym 200.

```
Grid search results:
                                                                 params  mean_test_score
0  {'hidden_layer_sizes': 100, 'learning_rate_init': 0.001, 'solver': 'adam'}        0.939517
Prediction output:
 [7 2 1 0 4 1 4 9 5 9]
Confusion matrix:
 [[1 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0]
 [0 0 0 2 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 2 0]]
```

Rys. 19: Wynik predykcji dla solwera ADAM.

4.3.3. Różnice w prawdopodobieństwie w zależności od solwera.

```
Prediction probability for test set:
 [[ 0.00002662  0.00003702  0.01614552  0.06142898  0.00000024  0.00075036  0.         99.91937874  0.00022446  0.00200804]
 [ 1.27533335  0.01212787 78.12176017 16.15980383  0.00000112  4.13048228  0.22915594  0.00004937  0.07119703  0.00008905]
 [ 0.00010049 99.74704509  0.13008467  0.0212213   0.00944632  0.00220303  0.01371375  0.04723236  0.02564598  0.00330702]
 [99.82113169  0.00106843  0.00201656  0.00113428  0.00005629  0.0072854   0.15322169  0.01394927  0.00006244  0.00007396]
 [ 0.06997991  0.00120349  0.13900548  0.00213466 95.04031282  0.02582513  0.26927204  0.02123329  0.65334041  3.77769275]
 [ 0.00000402 99.87835421  0.00684019  0.01282958  0.026894    0.00048756  0.00080775  0.06153713  0.00774629  0.00449927]
 [ 0.00038773  0.0000646   0.00810287  0.00262652 95.96430952  0.01284951  0.00171607  2.83060464  0.11737542  1.06196312]
 [ 0.00008426  0.00059198  0.00151119  0.03934667  4.09069209  0.03737516  0.0000887   0.0486918   0.00690166 95.77471649]
 [ 1.18528279  0.00026188  0.03746763  0.00006811 15.85121658 10.95440246 71.70279988  0.00044918  0.01515188  0.25289961]
 [ 0.00377327  0.00141237  0.00048941  0.01111117 26.99073281  0.25876907  0.0019683  11.0559851   1.95744206 59.71831643]]

Best approximation:
 [99.91937874 78.12176017 99.74704509 99.82113169 95.04031282 99.87835421 95.96430952 95.77471649 71.70279988 59.71831643]
```

Rys. 20: Wynik prawdopodobieństw dla solwera ADAM.

```
Prediction probability for test set:
 [[ 0.02957081  0.00003698  0.02930322  0.0295294   0.00033967  0.00121225  0.00002471 99.8990993   0.00219972  0.00868395]
 [ 1.35253357 12.44409243 54.0213447  24.38395383  0.00282979  2.45673603  2.23516023  0.00339878  3.07967207  0.02027856]
 [ 0.00008888 99.93245354  0.02685838  0.02160436  0.00013068  0.00086862  0.00267991  0.00876848  0.0056872   0.00085996]
 [98.19404586  0.00268693  0.19912289  0.06309834  0.0149079   1.11965815  0.12502086  0.15256676  0.03857388  0.09031843]
 [ 0.00306072  0.00167335  0.0713553   0.0070541  98.7531154   0.02057389  0.1399037   0.08341633  0.1037346   0.81611261]
 [ 0.00021845 99.85849703  0.01542389  0.01274204  0.00285507  0.00327272  0.00339587  0.07096348  0.03002808  0.00260338]
 [ 0.0169374   0.00246414  0.00947463  0.0396225  91.50610984  1.08706806  0.16889545  0.13454984  6.85356331  0.18131483]
 [ 0.00311418  0.17027039  0.75890594  0.89649028  0.85896995  0.41614243  0.05182098  0.79009086  0.30740678 95.74678821]
 [ 3.31365855  0.13929743  1.37595862  0.0081645   2.71008004  5.48106486 86.79365611  0.01007127  0.10753957  0.06050903]
 [ 0.02140414  0.00145211  0.00855892  0.01486269  1.55540514  0.02132986  0.00666741  3.83785715  0.80385778 93.7286048 ]]

Best approximation:
 [99.8990993  54.0213447  99.93245354 98.19404586 98.7531154  99.85849703 91.50610984 95.74678821 86.79365611 93.7286048 ]
```

Rys. 21: Wynik prawdopodobieństw dla solwera SGD.

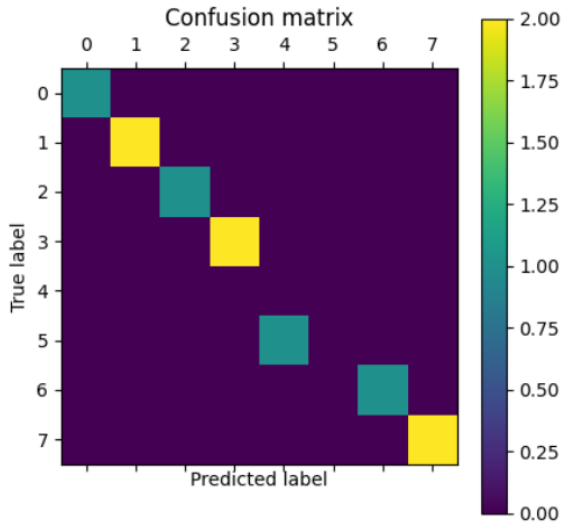
```
Prediction probability for test set:
 [[ 0.00310205  0.00015852  0.00503073  0.01122034  0.00154677  0.00187213  0.0000089  99.86448831  0.00998554  0.1025867 ]
 [ 0.22567161  0.63618986 53.18100073 14.5777206   0.00070222  5.25731434 25.28206711  0.00179724  0.83555968  0.00197661]
 [ 0.00013605 99.22948471  0.33279503  0.04538123  0.01596352  0.02329637  0.03171877  0.03984403  0.25299691  0.02838337]
 [99.65087528  0.00014055  0.02415975  0.01695419  0.00427318  0.19738216  0.07293394  0.00803778  0.00755066  0.01769251]
 [ 0.0032488   0.00044366  0.11709922  0.00128818 96.54332375  0.01654696  0.02012341  0.07425273  0.09388504  3.12978826]
 [ 0.00058388 99.15882932  0.43970989  0.0364375   0.01022357  0.00909651  0.00225258  0.11536678  0.19445549  0.03304449]
 [ 0.00048744  0.00130792  0.00562592  0.0086265  97.63838193  0.13429451  0.00329512  0.45911668  0.6244636   1.12440039]
 [ 0.00003579  0.00149781  0.00650617  0.00770042  3.0281085   0.01842044  0.00210402  0.08676044  0.07073426 96.77813214]
 [ 2.30619839  0.04164233 22.90404121  0.02340119  2.34281888  0.89001285 70.16246095  0.00514796  1.02375104  0.30052521]
 [ 0.00063696  0.00090184  0.00698275  0.00542619  1.93811102  0.00862842  0.00073098  1.0393442   0.09034669 96.90889097]]

Best approximation:
 [99.86448831 53.18100073 99.22948471 99.65087528 96.54332375 99.15882932 97.63838193 96.77813214 70.16246095 96.90889097]
```

Rys. 22: Wynik prawdopodobieństw dla solwera LBFGS.

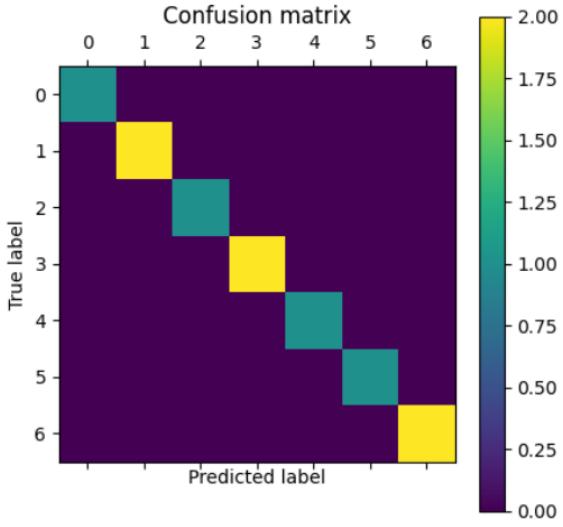
```
Prediction probability for test set:
 [[ 0.00008764  0.00000122  0.0001742   0.00029048  0.00000086  0.00000045  0.         99.99931478  0.00000073  0.00012962]
 [ 0.00153892  0.06687439 99.65035512  0.27381514  0.00000116  0.00399157  0.00170008  0.00000201  0.001712    0.0000096 ]
 [ 0.00000976 99.89335113  0.02234027  0.06667307  0.00000434  0.00038056  0.00003805  0.0151965   0.00197014  0.00003618]
 [99.73850203  0.00012706  0.05985529  0.01392436  0.00062925  0.04042097  0.05006978  0.01995545  0.00475778  0.07175803]
 [ 0.000728    0.00000001  0.0000643   0.00002436 99.81993567  0.00204026  0.00089126  0.0400055   0.00443448  0.13187616]
 [ 0.00007507 99.96736354  0.00580067  0.00644098  0.00000928  0.0006023   0.00043041  0.01800411  0.00126098  0.00001266]
 [ 0.00000217  0.00024868  0.00011701  0.0087974  97.08936968  0.02066424  0.00001361  0.57041861  2.11841433  0.19195428]
 [ 0.00010382  0.02043227  0.09189539  4.2881763   3.99697196  0.00781261  0.00003511  0.1402093   0.09197655 91.36238668]
 [ 0.00036721  0.00000876  0.0014594   0.00003128  0.05400681 59.22600732 40.71770714  0.00000064  0.00022209  0.00018935]
 [ 0.0111081   0.00023781  0.00223078  0.02147386  2.07013893  0.00791489  0.00001092 39.36228658  0.19774953 58.32684861]]

Best approximation:
 [99.99931478 99.65035512 99.89335113 99.73850203 99.81993567 99.96736354 97.08936968 91.36238668 59.22600732 58.32684861]
```

Rys. 23: Wynik prawdopodobieństw dla solwera ADAM dla 200 iteracji.

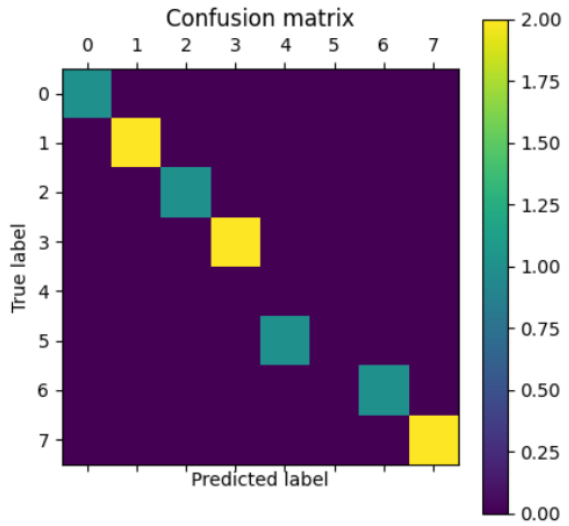
4.4. Porównanie macierzy pomyłek dla różnych solwerów.



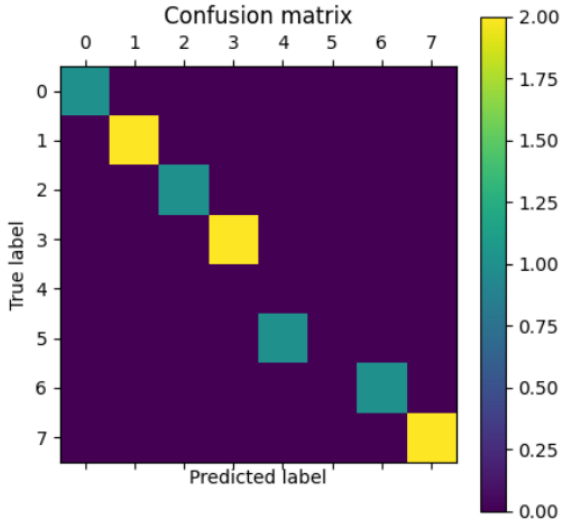
(a) Adam, max_iters = 20.



(b) Adam, max_iters = 200



(c) SGD, max_iters = 20.



(d) LBFGS, max_iters = 20

4.5. Dane Stellar.

Dla każdego solwera ustawiono parametr max_iter=50.

4.5.1. Wyniki funkcji GridSearch.

Grid search results:

	params	mean_test_score	rank_test_score
0	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.333400	14
1	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.918633	8
2	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.939733	6
3	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.457100	13
4	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.940900	5
5	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.949367	3
6	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.333333	15
7	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.923767	7
8	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.951067	1
9	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.562767	11
10	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.899233	9
11	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.950500	2
12	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.1, 'solver': 'adam'}	0.458800	12
13	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.01, 'solver': 'adam'}	0.857033	10
14	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.001, 'solver': 'adam'}	0.949333	4

Rys. 24: Skuteczność w zależności od parametrów dla solvera ADAM.

Grid search results:

	params	mean_test_score	rank_test_score
0	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.340800	13
1	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.367967	10
2	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.407533	6
3	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.352667	11
4	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.374100	9
5	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.430067	4
6	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.342433	12
7	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.376967	8
8	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.431933	3
9	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.339800	14
10	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.415733	5
11	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.444167	2
12	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.1, 'solver': 'sgd'}	0.336467	15
13	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.01, 'solver': 'sgd'}	0.394200	7
14	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.001, 'solver': 'sgd'}	0.445533	1

Rys. 25: Skuteczność w zależności od parametrów dla solvera SGD.

Grid search results:

	params	mean_test_score	rank_test_score
0	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}	0.618867	9
1	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}	0.623400	7
2	{'hidden_layer_sizes': 20, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}	0.610767	11
3	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}	0.628200	6
4	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}	0.641933	2
5	{'hidden_layer_sizes': 40, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}	0.622933	8
6	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}	0.647567	1
7	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}	0.637833	4
8	{'hidden_layer_sizes': 60, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}	0.608900	13
9	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}	0.598567	15
10	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}	0.609967	12
11	{'hidden_layer_sizes': 80, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}	0.607667	14
12	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.1, 'solver': 'lbfgs'}	0.616467	10
13	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.01, 'solver': 'lbfgs'}	0.634167	5
14	{'hidden_layer_sizes': 100, 'learning_rate_init': 0.001, 'solver': 'lbfgs'}	0.640833	3

Rys. 26: Skuteczność w zależności od parametrów dla solvera LBFGS.

4.5.2. Wyniki predykcji dla najlepszych parametrów wybranych przez GridSearch

Parametry:

- ilość neuronów w warstwie ukrytej: 60
- prędkość uczenia: 0.001

```
Prediction output:
 [0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 0 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 1 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 0 2 1 0
 2 2 2 2 2 2]
Confusion matrix:
 [[ 98   0   8]
 [  0 100   2]
 [  2   0  90]]
```

Rys. 27: Wynik predykcji dla solwera ADAM.

Parametry:

- ilość neuronów w warstwie ukrytej: 100
- prędkość uczenia: 0.001

```
Prediction output:
 [1 0 0 0 0 0 0 0 0 2 0 2 2 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 2 2
 0 0 0 0 2 0 2 0 0 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 2 2 0 1 0 0 0 0 0 0 0 0 2 0 0 0 0 0 2 0
 0 2 0 0 1 0 1 0 2 0 0 0 0 2 1 1 1 0 1 0 1 0 1 1 0 0 0 0 0 2 0 0 1 0 1 2 0 0 0 0 0 2 1 0 0 1 1 2 0
 0 0 0 2 0 2 0 0 0 0 2 0 0 0 0 2 1 0 0 1 0 0 0 0 2 2 1 1 0 0 0 1 0 0 1 0 1 0 2 0 1 0 1 0 0 1 2 1 0
 0 1 2 1 2 0 0 2 2 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 2 2 0 2 2 2 2 1 0 0 2 2 0 0 0 0 0 0 0 0 2 0 0 1 2
 2 0 2 2 2 2 2 2 2 0 0 0 2 2 2 2 2 0 0 2 1 2 1 0 0 2 2 0 2 0 0 2 0 2 0 0 0 2 0 0 0 2 0 0 0 2 2 0 2
 0 0 2 2 1 0]
Confusion matrix:
 [[79 58 53]
 [ 5 27  7]
 [16 15 40]]
```

Rys. 28: Wynik predykcji dla solwera SGD.

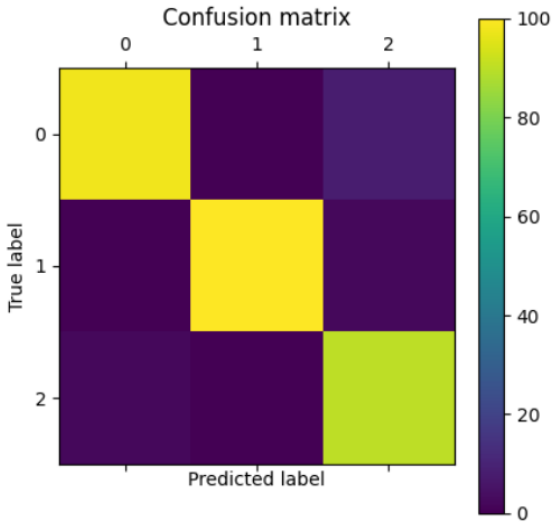
Parametry:

- ilość neuronów w warstwie ukrytej: 60
- prędkość uczenia: 0.1

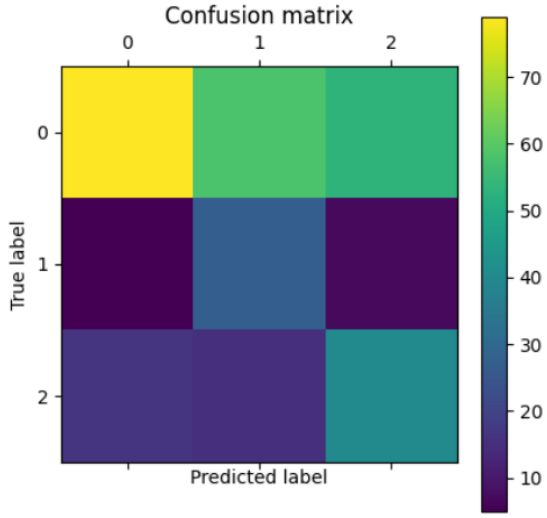
```
Prediction output:
 [0 0 0 0 0 0 0 0 0 0 1 2 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 2 1 0 1 2 0 0 2 1 1 0 0 1 1 0 1 2 0 1 1 1 1
 0 0 0 0 0 1 0 1 0 0 0 2 0 0 0 0 0 2 0 0 1 2 2 0 0 2 2 1 1 0 0 0 0 1 0 0 1 0 2 1 2 0 1 0 0 1 1 0 0
 0 1 0 2 0 0 1 1 1 1 1 2 0 2 0 2 1 0 1 1 1 2 0 0 1 2 0 1 0 1 1 0 1 0 1 0 1 0 0 0 2 1 1 1 1 1 0 0 1
 2 2 1 0 1 1 1 1 1 0 0 2 0 1 1 1 0 1 0 1 1 1 1 2 0 0 0 1 1 1 2 1 0 1 1 0 2 1 1 0 1 1 2 2 1 1 2 1 1
 0 1 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 1 2 2 2 2 2 0 2 2 2 2 2 2 0 2 2 2 2 2
 2 2 2 0 2 2 2 1 2 2 2 2 2 0 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 0 2 2 2 2 2 2 2 2 2 2 2 2 1 2
 2 2 2 2 2 2]
Confusion matrix:
 [[60 32  7]
 [27 51  6]
 [13 17 87]]
```

Rys. 29: Wynik predykcji dla solwera LBFGS.

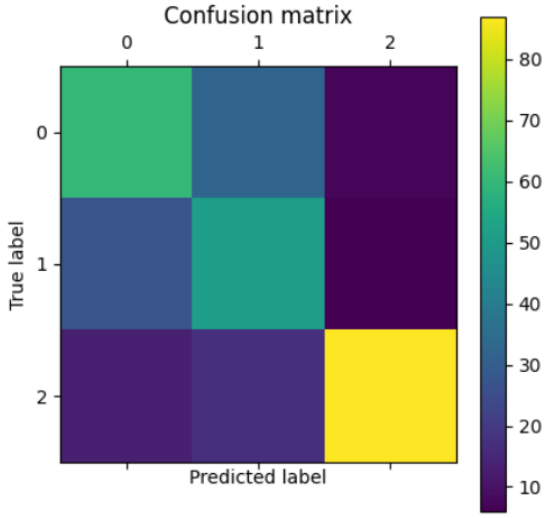
4.6. Porównanie macierzy pomyłek dla różnych solwerów.



Rys. 30: ADAM, max_iters = 50.



(a) SGD, max_iters = 50.



(b) LBFGS, max_iters = 50

5. Wnioski.

5.1. Dane Iris.

- Dla obu solwerów wykorzystujących gradient, kilka z modeli rozważanych w funkcji GridSearch osiągnęło tę samą wartość mean_test_score. Dla solwerów ADAM i LBFGS, modele dla wszystkich parametrów osiągnięty wynik miał wartość powyżej 0.94 prawidłowych wykryć. Dla solwera SGD wyniki były znacznie gorsze dla prędkości uczenia równej 0.01.

- Przy mniejszej liczbie neuronów w warstwie ukrytej najskuteczniejszy okazał się solwer LBFGS - przy prędkości uczenia równej 0.1 osiągnął zgodność 0.975.

- Przy dużych ilościach neuronów, wszystkie algorytmy wykryły prawidłowo wszystkie klasy. Dla małej ilości neuronów najlepszy solwer LBFGS popełnił 2 błędy (na 10 próbek danego zestawu testowego).

- Pomimo, że najlepszy z modeli LBFGS osiągnął gorszy wynik testu niż dwa najlepsze modele pozostałych solwerów, prawdopodobieństwo z jakim przewidywał poszczególne klasy było bardzo bliskie 1.

5.2. Dane MNIST.

- Najlepszy wynik spośród trzech solwerów osiągnął solwer SGD: 0.92985 zgodności. Bardzo podobny wynik osiągnął solwer ADAM: 0.922083. Najgorzej wypadł solwer LBFGS: 0.899833.

- Jeśli chodzi jednak o przypadki, w których modele miały 20 neuronów w warstwie ukrytej oraz największą prędkość uczenia, najlepiej spośród solwerów wypadł właśnie LBFGS: 0.83, podczas gdy solwery gradientowe osiągały dokładność 0.23 i 0.31.

- W losowo wybranym zbiorze testowym, jako przedostatni element pojawiła się cyfra, która najprawdopodobniej będąc cyfrą "5", posiada wszystkie właściwości cyfry "6". Dla 20 iteracji, żaden z solwerów nie zdołał sklasyfikować przedostatniej cyfry poprawnie. Każdy z nich klasyfikował ją jako cyfrę "6". Z największą pewnością, zrobił do algorytm SGD. Najmniej pewny był solwer LBFGS.

  Wykonano dodatkowy test dla solwera ADAM, przy 200 iteracjach, by sprawdzić, czy ich ilość poprawi wynik. Po tej zmianie algorytm określił przedostatnią próbkę jako cyfrę "5" z prawdopodobieństwem ponad 59%. Jest to bardzo dobry wynik, ponieważ człowiek patrzący na tę cyfrę również nie jest w stanie z całą pewnością stwierdzić do jakiej klasy należy.

5.3. Dane Stellar.

- Wyniki otrzymane z testów na bazie obiektów kosmicznych o 8 parametrach znacznie różnią się wyników dla poprzednich zbiorów danych. Solwer ADAM w najlepszym przypadku uzyskał średnią trafność równą 95%, lecz wyniki pozostałych dwóch solwerów znacznie się pogorszyły - pomimo wzrostu dokładności, wraz ze zmniejszeniem prędkości uczenia, solwer SGD uzyskał maksymalną średnią celność równą $\approx 44{,}6\%$. Solwer LBGFS uzyskał w najlepszym wypadku zgodność równą 64,7%, dla prędkości uczenia równej 0.1 oraz dla 60 neuronów w warstwie ukrytej. Wnioskując po samych macierzach pomyłek, można zauważyć dużą zbieżność algorytmu ADAM, oraz kształtującą się zbieżność algorytmu LBFGS. Co ciekawe, wyniki algorytmu SGD ledwo powyżej 33% sugerują, że tak nauczony algorytm rozpoznawania obarczony jest dużym błędem, częściej wykrywa błędnie niż poprawnie, podobnie mógłby się sprawdzić klasyfikator losowy.