Politechnika Warszawska

Sprawozdanie projektu grupowego
Klasyfikacja cyfr pisanych odręcznie

Wydział: Elektryczny
Kierunek: Informatyka stosowana
Przedmiot: Podstawy reprezentacji i analizy danych, Laboratoria
Grupa nr.6, w skład wchodzą:
Edvin Suchodolskij 308919
Konrad Žilinski 308920
Mateusz Pietrzak 307373

Warszawa 2020.02.06

Spis treści:

1. Wprowadzenie:

Opis ogólny problemu

Istnieje bardzo dużo różnych wariantów napisania tej samej liczby. W czasach automatyzacji urządzenia powinny nauczyć się rozpoznawać ludzkie pismo, tym samym pozwalając zastąpić pracowników w powtarzalnych pracach, tym samym umożliwić cyfryzację. Program po przetworzeniu tej struktury danych powinien być w stanie odróżnić praktycznie dowolną cyfrę.

Dane

Struktura danych MNIST zawiera 60000 przykładów i 10.000 testowych zdjęć. Dana struktura jest chętnie używana do uczenia maszynowego oraz jego testowania. Zapisana jest w plikach „.csv" i każdy z nich zawiera 785 liczb. Pierwsza liczba jest cyfrą, którą program ma rozpoznać. Kolejne 784 liczb, od 0 do 255, wskazują na jasność pikseli obrazu tej liczby w skali szarości. Rozmiar obrazu wynosi 28 x 28 pikseli. MNIST jest popularną bazą danych stworzoną dla ludzi, którzy chcą spróbować swoich sił w analizie danych bez spędzania zbytecznych sił na ich formatowanie.

Dostępne rozwiązania:

Metody które znaleźliśmy w Internecie[1]:

- Linear Classifiers
- Boosted Stumps
- Non-Linear Classifiers
- Support vector machines (SVMs)
- Convolutional nets

Sprawdziliśmy poprawność odpowiedzi dla metod:

- Neural Network Multi-Layer-Perceptron (MLP)
- Decision Tree
- Gaussian process classification

[1] http://yann.lecun.com/exdb/mnist/

- Gaussian Naive Bayes
- K-nearest neighbors
- Nearest centroids

Wybraliśmy te modele ponieważ z ograniczeń czasowych nie moglibyśmy, nie bylibyśmy w stanie sprawdzić dogłębnie i zrozumieć parametrów wszystkich modeli. Wybrane zostały modele omawiane wcześniej na zajęciach z Podstaw Reprezentacji i Analizy Danych. Uwzględnione zostały także: Gaussian process i Neural Network, gdzie dla ostatnich dwóch kryterium była ich skuteczność.

Metoda testów

Zdecydowaliśmy wypisywać wszystkie parametry w liniach obok siebie sortowanych rosnąco według celności modelu. Pokazywanie tych danych na wykresie wymagało by przygotowania programu, który by mógł interpretować graficznie te dane. Ponieważ mamy jedną wartość liczbową oraz wiele wartości kategorycznych ukazanie tych danych na wykresie byłoby nieproduktywne. Wypisując te dane w linii kosztem estetyki zyskujemy na czytelności.

2. Analiza kolejnych modeli

Nearest Centroid

Ta metoda polega na minimalizacji odległości wektorów wartości atrybutów obiektów należących do danego klastra do pewnego punktu charakterystycznego klastra (zwanego jego środkiem lub centroidem), do którego obiekty zostały przyporządkowane. Przyporządkowanie danego obiektu do klastra odbywa się poprzez porównanie jego odległości do wszystkich centroidów. Metoda ta wymaga informacji o liczbie klastrów (grup). Początkowe ich położenia wybierane są losowo albo z użyciem specjalnego algorytmu.

Test no.1

Testowane wartości parametru „metric”:

- Hamming
- Manhattan
- Euclidean
- Minkowski
- Cosine

Testowane wartości parametru „shrink_threshold”: od 0 do 1 co 0.1.

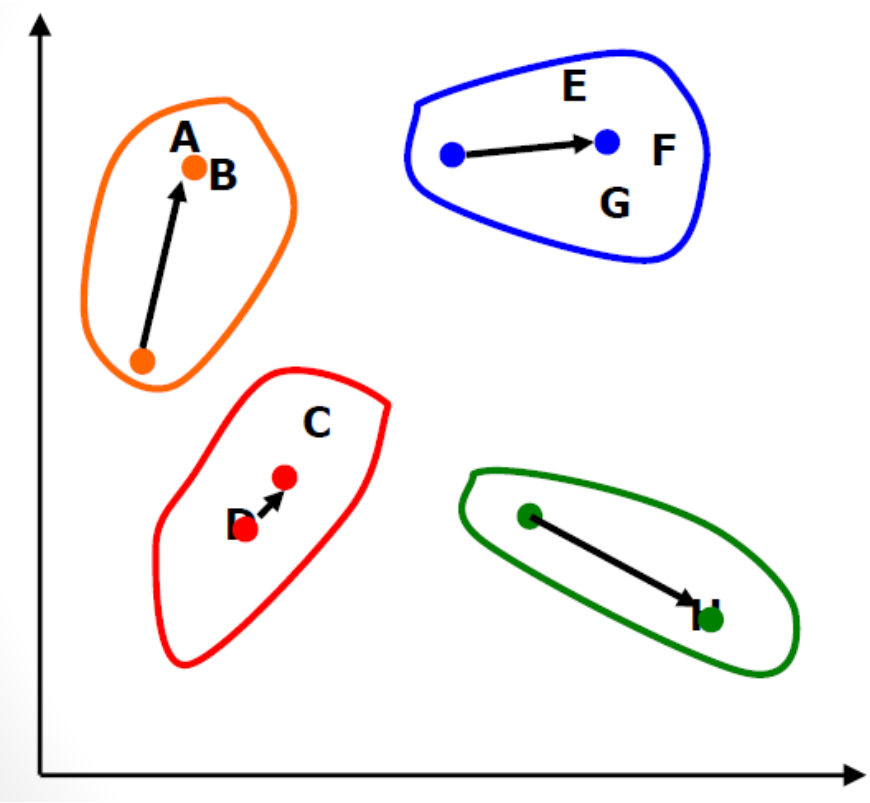
```
0.097 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 1}
0.097 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 0.9}
0.097 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 0.8}
0.097 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 0.7}
0.097 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 0.6}
0.097 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 0.5}
0.097 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 0.4}
0.097 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 0.3}
0.097 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 0.2}
0.097 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 0.1}
0.112 (+/-0.001) for {'metric': 'hamming', 'shrink_threshold': 0}
0.752 (+/-0.034) for {'metric': 'manhattan', 'shrink_threshold': 1}
0.752 (+/-0.033) for {'metric': 'manhattan', 'shrink_threshold': 0.6}
0.753 (+/-0.034) for {'metric': 'manhattan', 'shrink_threshold': 0.9}
0.753 (+/-0.034) for {'metric': 'manhattan', 'shrink_threshold': 0.5}
0.753 (+/-0.034) for {'metric': 'manhattan', 'shrink_threshold': 0.4}
0.753 (+/-0.033) for {'metric': 'manhattan', 'shrink_threshold': 0.8}
0.753 (+/-0.034) for {'metric': 'manhattan', 'shrink_threshold': 0}
0.753 (+/-0.034) for {'metric': 'manhattan', 'shrink_threshold': 0.7}
0.753 (+/-0.034) for {'metric': 'manhattan', 'shrink_threshold': 0.3}
0.753 (+/-0.034) for {'metric': 'manhattan', 'shrink_threshold': 0.2}
0.753 (+/-0.034) for {'metric': 'manhattan', 'shrink_threshold': 0.1}
0.819 (+/-0.026) for {'metric': 'euclidean', 'shrink_threshold': 1}
0.819 (+/-0.026) for {'metric': 'minkowski', 'shrink_threshold': 1}
0.819 (+/-0.027) for {'metric': 'euclidean', 'shrink_threshold': 0.3}
0.819 (+/-0.027) for {'metric': 'minkowski', 'shrink_threshold': 0.3}
0.819 (+/-0.027) for {'metric': 'euclidean', 'shrink_threshold': 0.9}
0.819 (+/-0.025) for {'metric': 'euclidean', 'shrink_threshold': 0}
0.819 (+/-0.027) for {'metric': 'minkowski', 'shrink_threshold': 0.9}
0.819 (+/-0.025) for {'metric': 'minkowski', 'shrink_threshold': 0}
0.819 (+/-0.026) for {'metric': 'euclidean', 'shrink_threshold': 0.1}
0.819 (+/-0.026) for {'metric': 'minkowski', 'shrink_threshold': 0.1}
0.819 (+/-0.027) for {'metric': 'euclidean', 'shrink_threshold': 0.4}
0.819 (+/-0.027) for {'metric': 'minkowski', 'shrink_threshold': 0.4}
0.820 (+/-0.026) for {'metric': 'euclidean', 'shrink_threshold': 0.2}
0.820 (+/-0.026) for {'metric': 'minkowski', 'shrink_threshold': 0.2}
0.820 (+/-0.028) for {'metric': 'euclidean', 'shrink_threshold': 0.7}
0.820 (+/-0.028) for {'metric': 'minkowski', 'shrink_threshold': 0.7}
0.820 (+/-0.027) for {'metric': 'euclidean', 'shrink_threshold': 0.8}
0.820 (+/-0.027) for {'metric': 'minkowski', 'shrink_threshold': 0.8}
0.820 (+/-0.027) for {'metric': 'euclidean', 'shrink_threshold': 0.6}
0.820 (+/-0.027) for {'metric': 'euclidean', 'shrink_threshold': 0.5}
0.820 (+/-0.027) for {'metric': 'minkowski', 'shrink_threshold': 0.6}
0.820 (+/-0.027) for {'metric': 'minkowski', 'shrink_threshold': 0.5}
0.824 (+/-0.034) for {'metric': 'cosine', 'shrink_threshold': 0.9}
0.824 (+/-0.034) for {'metric': 'cosine', 'shrink_threshold': 0.8}
0.824 (+/-0.033) for {'metric': 'cosine', 'shrink_threshold': 0.7}
0.825 (+/-0.033) for {'metric': 'cosine', 'shrink_threshold': 0.6}
0.825 (+/-0.034) for {'metric': 'cosine', 'shrink_threshold': 1}
0.825 (+/-0.031) for {'metric': 'cosine', 'shrink_threshold': 0.1}
0.825 (+/-0.032) for {'metric': 'cosine', 'shrink_threshold': 0}
0.825 (+/-0.032) for {'metric': 'cosine', 'shrink_threshold': 0.2}
0.825 (+/-0.033) for {'metric': 'cosine', 'shrink_threshold': 0.3}
0.825 (+/-0.032) for {'metric': 'cosine', 'shrink_threshold': 0.5}
0.826 (+/-0.031) for {'metric': 'cosine', 'shrink_threshold': 0.4}
```

Widzimy, że najlepszą wartością parametru „metric” jest cosine.

Natomiast wartości parametru „shrink_threschold” maja znikomy wpływ na jakość modelu.

Parametr „metric” - metryka używana podczas obliczania odległości między wystąpieniami w szyku elementów. Centroidy próbek odpowiadających każdej klasie to punkt, od którego suma odległości (zgodnie z metryką) wszystkich próbek należących do tej konkretnej klasy jest minimalizowana.

Parametr „shrink_threshold” – próg pomniejszania centroidu w celu zmniejszenia ilości elementów.



K-nearest neighbors

Zasadą działania metod najbliższego sąsiada jest poszukiwanie najbliższego sąsiada dla nowego obiektu o nieznanej klasie, wśród obiektów znajdujących się w zbiorze uczącym. Klasa, do której najbliższy sąsiad przynależy jest przypisywana klasyfikowanemu obiektowi.

Klasyfikator „k-najbliższych sąsiadów” jest uogólnieniem klasyfikatora najbliższego sąsiada. W jego przypadku, przynależność klasyfikowanego obiektu do klasy określana jest na podstawie znanych klas do których należy ustalona liczba „k” najbliższych sąsiadów. Klasa wynikowa odpowiada klasie dominującej w zbiorze „k-najbliższych sąsiadów".

Test no.1

Testowanie wartości parametru „algorithm”:

- Ball_tree
- Auto
- Brute
- Kd_tree

Testowanie wartosci parametru „metric”:

- Hamming
- Manhattan
- Euclidean
- Minkowski
- Cosine

Testowanie wartosci parametru „weights”:

- Uniform
- Distance

```
0.705 (+/-0.039) for {'algorithm': 'ball_tree', 'metric': 'hamming', 'n_jobs': 4, 'weights': 'uniform'}
0.706 (+/-0.040) for {'algorithm': 'auto', 'metric': 'hamming', 'n_jobs': 4, 'weights': 'uniform'}
0.706 (+/-0.040) for {'algorithm': 'brute', 'metric': 'hamming', 'n_jobs': 4, 'weights': 'uniform'}
0.718 (+/-0.034) for {'algorithm': 'ball_tree', 'metric': 'hamming', 'n_jobs': 4, 'weights': 'distance'}
0.719 (+/-0.032) for {'algorithm': 'auto', 'metric': 'hamming', 'n_jobs': 4, 'weights': 'distance'}
0.719 (+/-0.032) for {'algorithm': 'brute', 'metric': 'hamming', 'n_jobs': 4, 'weights': 'distance'}
0.936 (+/-0.009) for {'algorithm': 'auto', 'metric': 'manhattan', 'n_jobs': 4, 'weights': 'uniform'}
0.936 (+/-0.009) for {'algorithm': 'ball_tree', 'metric': 'manhattan', 'n_jobs': 4, 'weights': 'uniform'}
0.936 (+/-0.009) for {'algorithm': 'kd_tree', 'metric': 'manhattan', 'n_jobs': 4, 'weights': 'uniform'}
0.936 (+/-0.009) for {'algorithm': 'brute', 'metric': 'manhattan', 'n_jobs': 4, 'weights': 'uniform'}
0.940 (+/-0.011) for {'algorithm': 'auto', 'metric': 'manhattan', 'n_jobs': 4, 'weights': 'distance'}
0.940 (+/-0.011) for {'algorithm': 'ball_tree', 'metric': 'manhattan', 'n_jobs': 4, 'weights': 'distance'}
0.940 (+/-0.011) for {'algorithm': 'kd_tree', 'metric': 'manhattan', 'n_jobs': 4, 'weights': 'distance'}
0.940 (+/-0.011) for {'algorithm': 'brute', 'metric': 'manhattan', 'n_jobs': 4, 'weights': 'distance'}
0.945 (+/-0.008) for {'algorithm': 'auto', 'metric': 'euclidean', 'n_jobs': 4, 'weights': 'uniform'}
0.945 (+/-0.008) for {'algorithm': 'auto', 'metric': 'minkowski', 'n_jobs': 4, 'weights': 'uniform'}
0.945 (+/-0.008) for {'algorithm': 'ball_tree', 'metric': 'euclidean', 'n_jobs': 4, 'weights': 'uniform'}
0.945 (+/-0.008) for {'algorithm': 'ball_tree', 'metric': 'minkowski', 'n_jobs': 4, 'weights': 'uniform'}
0.945 (+/-0.008) for {'algorithm': 'kd_tree', 'metric': 'minkowski', 'n_jobs': 4, 'weights': 'uniform'}
0.945 (+/-0.008) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'weights': 'uniform'}
0.949 (+/-0.011) for {'algorithm': 'auto', 'metric': 'euclidean', 'n_jobs': 4, 'weights': 'distance'}
0.949 (+/-0.011) for {'algorithm': 'auto', 'metric': 'minkowski', 'n_jobs': 4, 'weights': 'distance'}
0.949 (+/-0.011) for {'algorithm': 'ball_tree', 'metric': 'euclidean', 'n_jobs': 4, 'weights': 'distance'}
0.954 (+/-0.011) for {'algorithm': 'auto', 'metric': 'cosine', 'n_jobs': 4, 'weights': 'uniform'}
0.957 (+/-0.010) for {'algorithm': 'auto', 'metric': 'cosine', 'n_jobs': 4, 'weights': 'distance'}
nan (+/-nan) for {'algorithm': 'ball_tree', 'metric': 'cosine', 'n_jobs': 4, 'weights': 'uniform'}
nan (+/-nan) for {'algorithm': 'ball_tree', 'metric': 'cosine', 'n_jobs': 4, 'weights': 'distance'}
0.949 (+/-0.011) for {'algorithm': 'ball_tree', 'metric': 'minkowski', 'n_jobs': 4, 'weights': 'distance'}
nan (+/-nan) for {'algorithm': 'kd_tree', 'metric': 'cosine', 'n_jobs': 4, 'weights': 'uniform'}
nan (+/-nan) for {'algorithm': 'kd_tree', 'metric': 'cosine', 'n_jobs': 4, 'weights': 'distance'}
nan (+/-nan) for {'algorithm': 'kd_tree', 'metric': 'hamming', 'n_jobs': 4, 'weights': 'uniform'}
nan (+/-nan) for {'algorithm': 'kd_tree', 'metric': 'hamming', 'n_jobs': 4, 'weights': 'distance'}
0.945 (+/-0.008) for {'algorithm': 'kd_tree', 'metric': 'euclidean', 'n_jobs': 4, 'weights': 'uniform'}
0.945 (+/-0.008) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'weights': 'uniform'}
0.949 (+/-0.011) for {'algorithm': 'kd_tree', 'metric': 'euclidean', 'n_jobs': 4, 'weights': 'distance'}
0.949 (+/-0.011) for {'algorithm': 'kd_tree', 'metric': 'minkowski', 'n_jobs': 4, 'weights': 'distance'}
0.949 (+/-0.011) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'weights': 'distance'}
0.949 (+/-0.011) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'weights': 'distance'}
0.954 (+/-0.011) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'weights': 'uniform'}
0.957 (+/-0.010) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'weights': 'distance'}
```

Dla takich kombinacji parametrów programowi nie udało się stworzyć modelu.

Wartość parametru „algorithm” „brute” skutkuje najlepszymi rezultatami.

Test no.2

```
0.935 (+/-0.015) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 2, 'weights': 'uniform'}
0.935 (+/-0.015) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 2, 'weights': 'uniform'}
0.943 (+/-0.012) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 6, 'weights': 'uniform'}
0.943 (+/-0.012) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 6, 'weights': 'uniform'}
0.943 (+/-0.010) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 4, 'weights': 'uniform'}
0.943 (+/-0.010) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 4, 'weights': 'uniform'}
0.945 (+/-0.008) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 5, 'weights': 'uniform'}
0.945 (+/-0.008) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 5, 'weights': 'uniform'}
0.948 (+/-0.016) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 3, 'weights': 'uniform'}
0.948 (+/-0.016) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 3, 'weights': 'uniform'}
0.949 (+/-0.011) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 5, 'weights': 'distance'}
0.949 (+/-0.011) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 5, 'weights': 'distance'}
0.949 (+/-0.016) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 1, 'weights': 'uniform'}
0.949 (+/-0.016) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 1, 'weights': 'distance'}
0.949 (+/-0.016) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 2, 'weights': 'distance'}
0.949 (+/-0.016) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 1, 'weights': 'uniform'}
0.949 (+/-0.016) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 1, 'weights': 'distance'}
0.949 (+/-0.016) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 2, 'weights': 'distance'}
0.950 (+/-0.016) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 3, 'weights': 'distance'}
0.950 (+/-0.016) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 3, 'weights': 'distance'}
0.951 (+/-0.014) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 2, 'weights': 'uniform'}
0.951 (+/-0.011) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 6, 'weights': 'distance'}
0.951 (+/-0.011) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 6, 'weights': 'distance'}
0.952 (+/-0.015) for {'algorithm': 'brute', 'metric': 'euclidean', 'n_jobs': 4, 'n_neighbors': 4, 'weights': 'distance'}
0.952 (+/-0.015) for {'algorithm': 'brute', 'metric': 'minkowski', 'n_jobs': 4, 'n_neighbors': 4, 'weights': 'distance'}
0.953 (+/-0.010) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 6, 'weights': 'uniform'}
0.954 (+/-0.011) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 5, 'weights': 'uniform'}
0.955 (+/-0.012) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 3, 'weights': 'uniform'}
0.955 (+/-0.013) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 4, 'weights': 'uniform'}
0.956 (+/-0.008) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 1, 'weights': 'uniform'}
0.956 (+/-0.008) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 1, 'weights': 'distance'}
0.956 (+/-0.008) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 2, 'weights': 'distance'}
0.957 (+/-0.010) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 5, 'weights': 'distance'}
0.957 (+/-0.011) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 6, 'weights': 'distance'}
0.957 (+/-0.013) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 3, 'weights': 'distance'}
0.959 (+/-0.014) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 4, 'weights': 'distance'}
```

Najlepsze wyniki są dla parametru „metric” o wartości „cosine”

Wartość „distance” dla parametru „weights” generuje najlepsze modele.

Test no.3

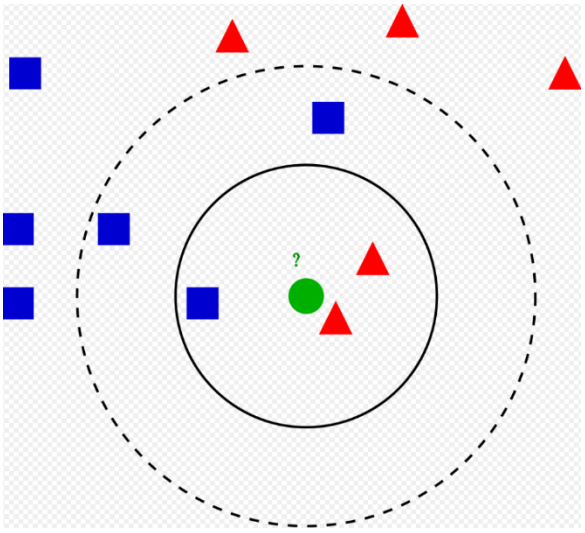
```
0.944 (+/-0.014) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 19, 'weights': 'distance'}
0.945 (+/-0.013) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 17, 'weights': 'distance'}
0.945 (+/-0.012) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 18, 'weights': 'distance'}
0.947 (+/-0.013) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 16, 'weights': 'distance'}
0.947 (+/-0.014) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 15, 'weights': 'distance'}
0.948 (+/-0.012) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 13, 'weights': 'distance'}
0.948 (+/-0.014) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 14, 'weights': 'distance'}
0.949 (+/-0.012) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 11, 'weights': 'distance'}
0.949 (+/-0.011) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 12, 'weights': 'distance'}
0.951 (+/-0.008) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 10, 'weights': 'distance'}
0.952 (+/-0.007) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 9, 'weights': 'distance'}
0.954 (+/-0.011) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 7, 'weights': 'distance'}
0.955 (+/-0.010) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 8, 'weights': 'distance'}
0.956 (+/-0.008) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 1, 'weights': 'distance'}
0.956 (+/-0.008) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 2, 'weights': 'distance'}
0.957 (+/-0.010) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 5, 'weights': 'distance'}
0.957 (+/-0.011) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 6, 'weights': 'distance'}
0.957 (+/-0.013) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 3, 'weights': 'distance'}
0.959 (+/-0.014) for {'algorithm': 'brute', 'metric': 'cosine', 'n_jobs': 4, 'n_neighbors': 4, 'weights': 'distance'}
```

„n_neighbors” z wartością „4” to najlepszy model

Parametr „algorithm” – jaki algorytm zostanie użyty do obliczenia najbliższych sąsiadów.

Parametr „metric” – metryka odległości dla drzewa.

Parametr „weight” – funkcja wagi używana w prognozowaniu.



Gaussian Naive Bayes

Przynależność obiektu do poszczególnych klas jest określana przy pomocy funkcji dyskryminacyjnych. i-ta funkcja dyskryminacyjna dla obiektu o wektorze atrybutów opisujących jest w tym przypadku tożsama prawdopodobieństwu warunkowemu przynależności obiektu do i-tej klasy pod warunkiem posiadania przez obiekt konkretnych cech. Wygodnym założeniem jest brak zależności między poszczególnymi atrybutami opisującymi. Dzięki niemu można przyjąć, że zdarzenia losowe polegające na posiadaniu przez obiekt konkretnych wartości poszczególnych atrybutów są od siebie niezależne.

Test no.1

Testowane wartości parametru „var_smoothing“.

```
0.153 (+/-0.036) for {'var_smoothing': 1000}
0.562 (+/-0.073) for {'var_smoothing': 100}
0.583 (+/-0.042) for {'var_smoothing': 1e-13}
0.590 (+/-0.049) for {'var_smoothing': 1e-12}
0.599 (+/-0.047) for {'var_smoothing': 1e-11}
0.609 (+/-0.054) for {'var_smoothing': 1e-10}
0.614 (+/-0.047) for {'var_smoothing': 1e-09}
0.625 (+/-0.055) for {'var_smoothing': 1e-08}
0.642 (+/-0.054) for {'var_smoothing': 1e-07}
0.642 (+/-0.054) for {'var_smoothing': 1e-07}
0.644 (+/-0.065) for {'var_smoothing': 10}
0.660 (+/-0.063) for {'var_smoothing': 1e-06}
0.683 (+/-0.081) for {'var_smoothing': 1e-05}
0.714 (+/-0.067) for {'var_smoothing': 0.0001}
0.739 (+/-0.071) for {'var_smoothing': 1}
0.751 (+/-0.091) for {'var_smoothing': 0.001}
0.804 (+/-0.082) for {'var_smoothing': 0.01}
0.809 (+/-0.086) for {'var_smoothing': 0.1}
```

Wartości powyżej i poniżej 0,1 parametru „var_smoothing” coraz bardziej pogarszają wynik.

Test no.2

```
0.707 (+/-0.068) for {'var_smoothing': 1.99}  0.722 (+/-0.072) for {'var_smoothing': 1.41}  0.745 (+/-0.071) for {'var_smoothing': 0.89}  0.788 (+/-0.073) for {'var_smoothing': 0.36}
0.708 (+/-0.067) for {'var_smoothing': 1.98}  0.722 (+/-0.065) for {'var_smoothing': 1.45}  0.745 (+/-0.071) for {'var_smoothing': 0.9}   0.788 (+/-0.073) for {'var_smoothing': 0.37}
0.709 (+/-0.065) for {'var_smoothing': 1.94}  0.723 (+/-0.070) for {'var_smoothing': 1.4}   0.745 (+/-0.071) for {'var_smoothing': 0.91}  0.792 (+/-0.069) for {'var_smoothing': 0.35}
0.709 (+/-0.065) for {'var_smoothing': 1.95}  0.723 (+/-0.069) for {'var_smoothing': 1.42}  0.749 (+/-0.071) for {'var_smoothing': 0.85}  0.793 (+/-0.074) for {'var_smoothing': 0.33}
0.709 (+/-0.065) for {'var_smoothing': 1.96}  0.723 (+/-0.069) for {'var_smoothing': 1.43}  0.749 (+/-0.071) for {'var_smoothing': 0.86}  0.794 (+/-0.072) for {'var_smoothing': 0.34}
0.709 (+/-0.065) for {'var_smoothing': 1.97}  0.723 (+/-0.069) for {'var_smoothing': 1.44}  0.749 (+/-0.071) for {'var_smoothing': 0.87}  0.796 (+/-0.072) for {'var_smoothing': 0.32}
0.711 (+/-0.066) for {'var_smoothing': 1.92}  0.724 (+/-0.069) for {'var_smoothing': 1.37}  0.749 (+/-0.071) for {'var_smoothing': 0.88}  0.796 (+/-0.078) for {'var_smoothing': 0.29}
0.711 (+/-0.066) for {'var_smoothing': 1.93}  0.724 (+/-0.069) for {'var_smoothing': 1.38}  0.750 (+/-0.070) for {'var_smoothing': 0.83}  0.797 (+/-0.075) for {'var_smoothing': 0.3}
0.712 (+/-0.065) for {'var_smoothing': 1.86}  0.724 (+/-0.069) for {'var_smoothing': 1.39}  0.750 (+/-0.070) for {'var_smoothing': 0.84}  0.797 (+/-0.075) for {'var_smoothing': 0.31}
0.712 (+/-0.065) for {'var_smoothing': 1.87}  0.726 (+/-0.072) for {'var_smoothing': 1.36}  0.751 (+/-0.071) for {'var_smoothing': 0.82}  0.798 (+/-0.078) for {'var_smoothing': 0.28}
0.712 (+/-0.065) for {'var_smoothing': 1.88}  0.727 (+/-0.069) for {'var_smoothing': 1.34}  0.752 (+/-0.069) for {'var_smoothing': 0.79}  0.799 (+/-0.077) for {'var_smoothing': 0.27}
0.712 (+/-0.065) for {'var_smoothing': 1.89}  0.727 (+/-0.069) for {'var_smoothing': 1.35}  0.752 (+/-0.069) for {'var_smoothing': 0.8}   0.803 (+/-0.079) for {'var_smoothing': 0.26}
0.712 (+/-0.065) for {'var_smoothing': 1.9}   0.728 (+/-0.069) for {'var_smoothing': 1.32}  0.752 (+/-0.069) for {'var_smoothing': 0.81}  0.804 (+/-0.082) for {'var_smoothing': 0.01}
0.712 (+/-0.065) for {'var_smoothing': 1.91}  0.728 (+/-0.069) for {'var_smoothing': 1.33}  0.753 (+/-0.066) for {'var_smoothing': 0.77}  0.804 (+/-0.082) for {'var_smoothing': 0.25}
0.713 (+/-0.065) for {'var_smoothing': 1.79}  0.730 (+/-0.068) for {'var_smoothing': 1.29}  0.753 (+/-0.066) for {'var_smoothing': 0.78}  0.807 (+/-0.084) for {'var_smoothing': 0.09}
0.713 (+/-0.065) for {'var_smoothing': 1.8}   0.730 (+/-0.068) for {'var_smoothing': 1.3}   0.755 (+/-0.071) for {'var_smoothing': 0.76}  0.809 (+/-0.086) for {'var_smoothing': 0.1}
0.713 (+/-0.065) for {'var_smoothing': 1.81}  0.730 (+/-0.068) for {'var_smoothing': 1.31}  0.759 (+/-0.068) for {'var_smoothing': 0.75}  0.809 (+/-0.083) for {'var_smoothing': 0.24}
0.713 (+/-0.065) for {'var_smoothing': 1.82}  0.731 (+/-0.066) for {'var_smoothing': 1.27}  0.762 (+/-0.067) for {'var_smoothing': 0.74}  0.810 (+/-0.081) for {'var_smoothing': 0.02}
0.713 (+/-0.065) for {'var_smoothing': 1.83}  0.731 (+/-0.066) for {'var_smoothing': 1.28}  0.763 (+/-0.069) for {'var_smoothing': 0.64}  0.810 (+/-0.084) for {'var_smoothing': 0.22}
0.713 (+/-0.065) for {'var_smoothing': 1.84}  0.732 (+/-0.066) for {'var_smoothing': 1.24}  0.763 (+/-0.069) for {'var_smoothing': 0.65}  0.810 (+/-0.084) for {'var_smoothing': 0.23}
0.713 (+/-0.065) for {'var_smoothing': 1.85}  0.732 (+/-0.066) for {'var_smoothing': 1.25}  0.763 (+/-0.069) for {'var_smoothing': 0.66}  0.811 (+/-0.089) for {'var_smoothing': 0.21}
0.714 (+/-0.065) for {'var_smoothing': 1.71}  0.732 (+/-0.066) for {'var_smoothing': 1.26}  0.763 (+/-0.069) for {'var_smoothing': 0.67}  0.811 (+/-0.082) for {'var_smoothing': 0.11}
0.714 (+/-0.065) for {'var_smoothing': 1.72}  0.734 (+/-0.068) for {'var_smoothing': 1.23}  0.763 (+/-0.069) for {'var_smoothing': 0.68}  0.812 (+/-0.068) for {'var_smoothing': 0.06}
0.714 (+/-0.065) for {'var_smoothing': 1.73}  0.735 (+/-0.066) for {'var_smoothing': 1.15}  0.763 (+/-0.069) for {'var_smoothing': 0.69}  0.812 (+/-0.078) for {'var_smoothing': 0.14}
0.714 (+/-0.065) for {'var_smoothing': 1.74}  0.735 (+/-0.066) for {'var_smoothing': 1.16}  0.763 (+/-0.069) for {'var_smoothing': 0.7}   0.812 (+/-0.084) for {'var_smoothing': 0.15}
0.714 (+/-0.065) for {'var_smoothing': 1.75}  0.735 (+/-0.066) for {'var_smoothing': 1.17}  0.763 (+/-0.069) for {'var_smoothing': 0.71}  0.812 (+/-0.088) for {'var_smoothing': 0.19}
0.714 (+/-0.065) for {'var_smoothing': 1.76}  0.735 (+/-0.066) for {'var_smoothing': 1.18}  0.763 (+/-0.069) for {'var_smoothing': 0.72}  0.812 (+/-0.087) for {'var_smoothing': 0.2}
0.714 (+/-0.065) for {'var_smoothing': 1.77}  0.735 (+/-0.066) for {'var_smoothing': 1.19}  0.763 (+/-0.069) for {'var_smoothing': 0.73}  0.813 (+/-0.074) for {'var_smoothing': 0.05}
0.714 (+/-0.065) for {'var_smoothing': 1.78}  0.735 (+/-0.066) for {'var_smoothing': 1.2}   0.764 (+/-0.069) for {'var_smoothing': 0.63}  0.813 (+/-0.078) for {'var_smoothing': 0.12}
0.715 (+/-0.062) for {'var_smoothing': 1.66}  0.735 (+/-0.066) for {'var_smoothing': 1.21}  0.765 (+/-0.071) for {'var_smoothing': 0.62}  0.814 (+/-0.083) for {'var_smoothing': 0.17}
0.715 (+/-0.062) for {'var_smoothing': 1.67}  0.735 (+/-0.066) for {'var_smoothing': 1.22}  0.766 (+/-0.069) for {'var_smoothing': 0.61}  0.814 (+/-0.083) for {'var_smoothing': 0.08}
0.715 (+/-0.062) for {'var_smoothing': 1.68}  0.736 (+/-0.067) for {'var_smoothing': 1.12}  0.767 (+/-0.070) for {'var_smoothing': 0.6}   0.814 (+/-0.073) for {'var_smoothing': 0.13}
0.715 (+/-0.062) for {'var_smoothing': 1.69}  0.736 (+/-0.067) for {'var_smoothing': 1.13}  0.769 (+/-0.070) for {'var_smoothing': 0.58}  0.814 (+/-0.086) for {'var_smoothing': 0.18}
0.715 (+/-0.062) for {'var_smoothing': 1.7}   0.736 (+/-0.067) for {'var_smoothing': 1.14}  0.769 (+/-0.070) for {'var_smoothing': 0.59}  0.815 (+/-0.085) for {'var_smoothing': 0.16}
0.717 (+/-0.063) for {'var_smoothing': 1.62}  0.737 (+/-0.064) for {'var_smoothing': 1.08}  0.772 (+/-0.074) for {'var_smoothing': 0.56}  0.816 (+/-0.079) for {'var_smoothing': 0.03}
0.717 (+/-0.063) for {'var_smoothing': 1.63}  0.737 (+/-0.064) for {'var_smoothing': 1.09}  0.772 (+/-0.074) for {'var_smoothing': 0.57}  0.816 (+/-0.079) for {'var_smoothing': 0.04}
0.717 (+/-0.063) for {'var_smoothing': 1.64}  0.737 (+/-0.064) for {'var_smoothing': 1.1}   0.774 (+/-0.072) for {'var_smoothing': 0.54}  0.816 (+/-0.074) for {'var_smoothing': 0.07}
0.717 (+/-0.063) for {'var_smoothing': 1.65}  0.737 (+/-0.064) for {'var_smoothing': 1.11}  0.774 (+/-0.072) for {'var_smoothing': 0.55}
0.718 (+/-0.064) for {'var_smoothing': 1.6}   0.738 (+/-0.065) for {'var_smoothing': 1.07}  0.776 (+/-0.071) for {'var_smoothing': 0.52}
0.718 (+/-0.064) for {'var_smoothing': 1.61}  0.739 (+/-0.064) for {'var_smoothing': 1.02}  0.776 (+/-0.066) for {'var_smoothing': 0.53}
0.719 (+/-0.065) for {'var_smoothing': 1.54}  0.739 (+/-0.064) for {'var_smoothing': 1.03}  0.777 (+/-0.070) for {'var_smoothing': 0.5}
0.719 (+/-0.065) for {'var_smoothing': 1.55}  0.739 (+/-0.064) for {'var_smoothing': 1.04}  0.777 (+/-0.070) for {'var_smoothing': 0.51}
0.719 (+/-0.065) for {'var_smoothing': 1.56}  0.739 (+/-0.064) for {'var_smoothing': 1.05}  0.778 (+/-0.072) for {'var_smoothing': 0.49}
0.719 (+/-0.065) for {'var_smoothing': 1.57}  0.739 (+/-0.064) for {'var_smoothing': 1.06}  0.780 (+/-0.069) for {'var_smoothing': 0.48}
0.719 (+/-0.065) for {'var_smoothing': 1.58}  0.739 (+/-0.071) for {'var_smoothing': 1.0}   0.781 (+/-0.072) for {'var_smoothing': 0.47}
0.719 (+/-0.065) for {'var_smoothing': 1.59}  0.739 (+/-0.071) for {'var_smoothing': 1.01}  0.782 (+/-0.075) for {'var_smoothing': 0.46}
0.720 (+/-0.064) for {'var_smoothing': 1.49}  0.740 (+/-0.072) for {'var_smoothing': 0.96}  0.783 (+/-0.073) for {'var_smoothing': 0.43}
0.720 (+/-0.064) for {'var_smoothing': 1.5}   0.740 (+/-0.072) for {'var_smoothing': 0.97}  0.783 (+/-0.072) for {'var_smoothing': 0.41}
0.720 (+/-0.064) for {'var_smoothing': 1.51}  0.740 (+/-0.072) for {'var_smoothing': 0.98}  0.783 (+/-0.072) for {'var_smoothing': 0.42}
0.720 (+/-0.064) for {'var_smoothing': 1.52}  0.740 (+/-0.072) for {'var_smoothing': 0.99}  0.783 (+/-0.069) for {'var_smoothing': 0.44}
0.720 (+/-0.064) for {'var_smoothing': 1.53}  0.741 (+/-0.071) for {'var_smoothing': 0.95}  0.783 (+/-0.071) for {'var_smoothing': 0.45}
0.721 (+/-0.061) for {'var_smoothing': 1.46}  0.743 (+/-0.071) for {'var_smoothing': 0.93}  0.785 (+/-0.068) for {'var_smoothing': 0.4}
0.721 (+/-0.061) for {'var_smoothing': 1.47}  0.743 (+/-0.071) for {'var_smoothing': 0.94}  0.786 (+/-0.071) for {'var_smoothing': 0.38}
0.721 (+/-0.061) for {'var_smoothing': 1.48}  0.744 (+/-0.069) for {'var_smoothing': 0.92}  0.786 (+/-0.071) for {'var_smoothing': 0.39}
```

Zmniejszanie wartości przynosi skutki aż do mniej więcej 0.25 gdy rezultaty zaczynają być chaotyczne.

„var_smoothing“ - część największej wariancji wszystkich cech, która jest dodawana do wariancji w celu zapewnienia stabilności obliczeń.

Gaussian process classification

Proces Gaussa jest procesem stochastycznym (zbiorem zmiennych losowych indeksowanych w czasie lub przestrzeni), tak że każdy skończony zbiór tych zmiennych losowych ma wielowymiarowy rozkład normalny, tj. Każda skończona ich kombinacja liniowa ma rozkład normalny.

Test no.1

Testowane wartości parametru „kernel”:

- 1**2 * RBF(length_scale=1)
- 1**2 * Matern(length_scale=1, nu=1.5)
- 1**2 * WhiteKernel(noise_level=1)
- 1**2 * RationalQuadratic(alpha=1, lenght_scale=1)

Testowanie wartości parametru „multi_class”:

- One_vs_rest
- One_vs_one

```
0.101 (+/-0.002) for {'kernel': 1**2 * RBF(length_scale=1), 'multi_class': 'one_vs_rest', 'n_restarts_optimizer': 0, 'optimizer': 'fmin_l_bfgs_b'}
0.101 (+/-0.002) for {'kernel': 1**2 * Matern(length_scale=1, nu=1.5), 'multi_class': 'one_vs_rest', 'n_restarts_optimizer': 0, 'optimizer': 'fmin_l_bfgs_b'}
0.101 (+/-0.002) for {'kernel': 1**2 * WhiteKernel(noise_level=1), 'multi_class': 'one_vs_rest', 'n_restarts_optimizer': 0, 'optimizer': 'fmin_l_bfgs_b'}
0.105 (+/-0.002) for {'kernel': 1**2 * RBF(length_scale=1), 'multi_class': 'one_vs_one', 'n_restarts_optimizer': 0, 'optimizer': 'fmin_l_bfgs_b'}
0.105 (+/-0.002) for {'kernel': 1**2 * WhiteKernel(noise_level=1), 'multi_class': 'one_vs_one', 'n_restarts_optimizer': 0, 'optimizer': 'fmin_l_bfgs_b'}
0.751 (+/-0.094) for {'kernel': 1**2 * RationalQuadratic(alpha=1, length_scale=1), 'multi_class': 'one_vs_one', 'n_restarts_optimizer': 0, 'optimizer': 'fmin_l_bfgs_b'}
0.872 (+/-0.004) for {'kernel': 1**2 * Matern(length_scale=1, nu=1.5), 'multi_class': 'one_vs_one', 'n_restarts_optimizer': 0, 'optimizer': 'fmin_l_bfgs_b'}
0.898 (+/-0.012) for {'kernel': 1**2 * RationalQuadratic(alpha=1, length_scale=1), 'multi_class': 'one_vs_rest', 'n_restarts_optimizer': 0, 'optimizer': 'fmin_l_bfgs_b'}
```

Dla jądra „Matern” ważne żeby „multi_class” był „one_vs_one”.

Obserwujemy mniejszy wpływ parametru „multi_class” dla jądra „RationalQuadratic”. Ten parametr jest kluczowy dla dobrych wyników modelu.

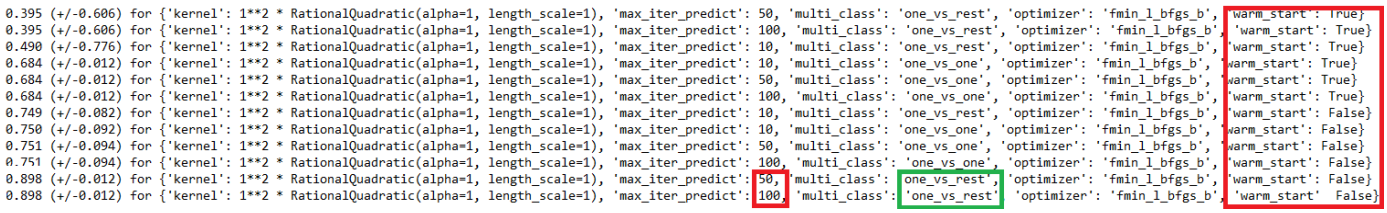
Test no.2

Testowanie wartości parametru „max_iter_predict”:

- 10
- 50
- 100

Testowanie wartości parametru „warm_start”:

- True
- False



Najlepsze wyniki uzyskuje model z parametrem „multi_class” z wartościa „one_vs_rest”

Parametr „warm_start” u najlepszych modeli jest ustawiony na „false”

„kernel” - Jądro określające funkcję kowariancji GP.

„multi_class” - Określa, w jaki sposób są obsługiwane problemy klasyfikacji wieloklasowej.

„max_iter_predict” - Maksymalna liczba iteracji w metodzie Newtona aproksymacji późniejszej podczas przewidywania.

„warm_start” - Jeśli włączone są ciepłe starty, rozwiązanie ostatniej iteracji Newtona w przybliżeniu Laplace'a trybu późniejszego jest używane jako inicjalizacja dla następnego wywołania _posterior_mode ().

Skuteczność tej metody jest nawet dość wysoka, jednak ogromnie duży czas jaki potrzebuje do trenowania jest nie do przyjęcia, co poskutkowało zaprzestaniem dalszych testów z nią.

Decision tree clasifier

Omówmy znaczenie atrybutów opisujących na podstawie zbioru Tytanic Data Set który zawiera informacje o pasażerach, takie jak klasa podróżna czy też wiek. Celem klasyfikacji jest odgadnięcie czy pasażer przeżył.

W tym zbiorze w korzeniu drzewa atrybutem opisującym jest płeć. Ponieważ najpierw ratowane kobiety to przeżyło ich znacznie więcej. Jest to także bardzo dobry podział pasażerów na dwie grupy.

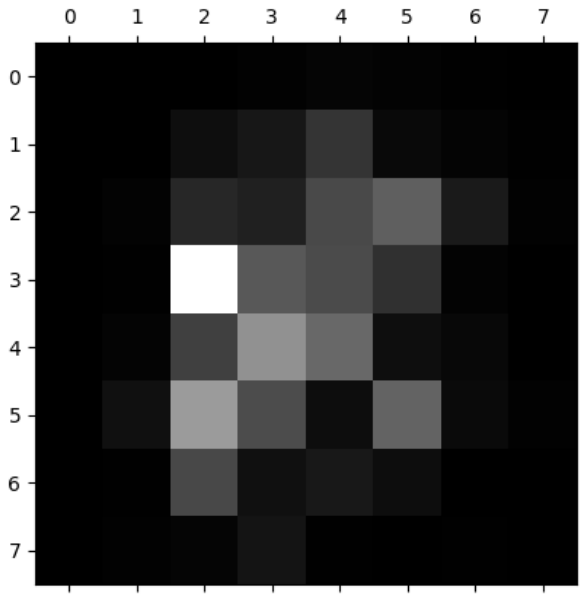
Dzięki temu podziałowi zyskujemy jak najwięcej informacji.

Drzewa decyzyjne są strukturą grafowa przedstawiającą zależności między atrybutami obiektów. Dzięki hierarchicznej reprezentacji tych zależności drzewo nie tylko jest klasyfikatorem, ale także umożliwia analizę istotności poszczególnych atrybutów w klasyfikacji konkretnego zbioru danych.

Jak to wygląda w drzewie decyzyjnym którego atrybutami są pojedyncze piksele? Szukamy piksela, który często występuje w jednej z grup liczb.

X[26] <= 0.5
entropy = 3.32
samples = 7500
value = [726, 843, 755, 757, 747, 691, 726, 766, 729, 760]

Poniżej zamieszczona jest grafika obrazująca wagę atrybutów. Widzimy na niej biały piksel w 2 kolumnie 3 rzędu. Nasze dane przechowujemy w liście o długości 64(8x8) indeksowanej od 0, dlatego element o indeksie 26 to nasz biały kwadrat. Ma największą wagę, ponieważ dokonuje największego podziału danych. Możemy więc powiedzieć, że ten piksel jest różnicą pomiędzy dwoma połowami naszych danych. Co za tym idzie, ten wykres przedstawia nam najważniejsze piksele obrazu.



Test no.1

Testowanie wartości parametru „criterion”:

- Entropy
- Gini

```
0.179 (+/-0.006) for {'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 8, 'splitter': 'random'}
0.182 (+/-0.052) for {'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 8, 'splitter': 'random'}
0.196 (+/-0.052) for {'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 4, 'splitter': 'random'}
0.197 (+/-0.038) for {'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 2, 'splitter': 'random'}
0.200 (+/-0.004) for {'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 6, 'splitter': 'random'}
0.205 (+/-0.006) for {'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 10, 'splitter': 'random'}
0.206 (+/-0.004) for {'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 4, 'splitter': 'random'}
0.208 (+/-0.012) for {'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 2, 'splitter': 'random'}
0.216 (+/-0.008) for {'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 2, 'splitter': 'best'}
0.216 (+/-0.008) for {'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 4, 'splitter': 'best'}
0.216 (+/-0.008) for {'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 6, 'splitter': 'best'}
0.216 (+/-0.008) for {'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 8, 'splitter': 'best'}
0.216 (+/-0.008) for {'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 10, 'splitter': 'best'}
0.217 (+/-0.006) for {'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 10, 'splitter': 'random'}
0.666 (+/-0.044) for {'criterion': 'entropy', 'max_depth': 50, 'min_samples_split': 8, 'splitter': 'best'}
0.666 (+/-0.028) for {'criterion': 'entropy', 'max_depth': 100000, 'min_samples_split': 2, 'splitter': 'best'}
0.667 (+/-0.026) for {'criterion': 'entropy', 'max_depth': 500, 'min_samples_split': 4, 'splitter': 'best'}
0.668 (+/-0.012) for {'criterion': 'gini', 'max_depth': 500, 'min_samples_split': 2, 'splitter': 'best'}
0.670 (+/-0.008) for {'criterion': 'entropy', 'max_depth': 5000, 'min_samples_split': 4, 'splitter': 'best'}
0.670 (+/-0.020) for {'criterion': 'gini', 'max_depth': 1000, 'min_samples_split': 4, 'splitter': 'best'}
0.672 (+/-0.040) for {'criterion': 'entropy', 'max_depth': 5000, 'min_samples_split': 10, 'splitter': 'best'}
0.672 (+/-0.036) for {'criterion': 'entropy', 'max_depth': 100000, 'min_samples_split': 6, 'splitter': 'best'}
0.672 (+/-0.048) for {'criterion': 'gini', 'max_depth': 5000, 'min_samples_split': 6, 'splitter': 'random'}
0.672 (+/-0.000) for {'criterion': 'entropy', 'max_depth': 100, 'min_samples_split': 2, 'splitter': 'best'}
0.673 (+/-0.010) for {'criterion': 'entropy', 'max_depth': 500, 'min_samples_split': 6, 'splitter': 'random'}
0.674 (+/-0.024) for {'criterion': 'entropy', 'max_depth': 50, 'min_samples_split': 4, 'splitter': 'best'}
0.675 (+/-0.042) for {'criterion': 'entropy', 'max_depth': 5000, 'min_samples_split': 2, 'splitter': 'best'}
```

Najgorsze modele powstają przy parametrze „criterion” dla wartości „gini”

Natomiast najlepsze dla „entropy”

Test no.2

Testowane wartości „splitter”:

- Random
- Best

```
0.790 (+/-0.035) for {'criterion': 'entropy', 'max_depth': 5000, 'min_samples_split': 8, 'splitter': 'random'}
0.791 (+/-0.022) for {'criterion': 'entropy', 'max_depth': 1000, 'min_samples_split': 9, 'splitter': 'random'}
0.792 (+/-0.028) for {'criterion': 'entropy', 'max_depth': 1000, 'min_samples_split': 10, 'splitter': 'random'}
0.793 (+/-0.029) for {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 9, 'splitter': 'random'}
0.793 (+/-0.028) for {'criterion': 'entropy', 'max_depth': 2000, 'min_samples_split': 7, 'splitter': 'random'}
0.793 (+/-0.023) for {'criterion': 'entropy', 'max_depth': 100, 'min_samples_split': 3, 'splitter': 'random'}
0.794 (+/-0.027) for {'criterion': 'entropy', 'max_depth': 20, 'min_samples_split': 4, 'splitter': 'random'}
0.795 (+/-0.037) for {'criterion': 'entropy', 'max_depth': 5000, 'min_samples_split': 7, 'splitter': 'random'}
0.796 (+/-0.025) for {'criterion': 'entropy', 'max_depth': 50, 'min_samples_split': 7, 'splitter': 'random'}
0.796 (+/-0.031) for {'criterion': 'entropy', 'max_depth': 50, 'min_samples_split': 4, 'splitter': 'random'}
0.796 (+/-0.027) for {'criterion': 'entropy', 'max_depth': 200, 'min_samples_split': 10, 'splitter': 'random'}
0.796 (+/-0.039) for {'criterion': 'entropy', 'max_depth': 500, 'min_samples_split': 6, 'splitter': 'random'}
0.797 (+/-0.033) for {'criterion': 'entropy', 'max_depth': 100, 'min_samples_split': 9, 'splitter': 'random'}
0.797 (+/-0.039) for {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 4, 'splitter': 'random'}
0.797 (+/-0.032) for {'criterion': 'entropy', 'max_depth': 500, 'min_samples_split': 10, 'splitter': 'random'}
0.797 (+/-0.033) for {'criterion': 'entropy', 'max_depth': 100000, 'min_samples_split': 9, 'splitter': 'random'
0.798 (+/-0.028) for {'criterion': 'entropy', 'max_depth': 100000, 'min_samples_split': 8, 'splitter': 'random'
```

Parametr „splitter” z wartością „random” nie jest najlepszym generuje najlepszego modelu

```
0.807 (+/-0.038) for {'criterion': 'entropy', 'max_depth': 20, 'min_samples_split': 2, 'splitter': 'best'}
0.807 (+/-0.033) for {'criterion': 'entropy', 'max_depth': 5000, 'min_samples_split': 3, 'splitter': 'best'}
0.807 (+/-0.032) for {'criterion': 'entropy', 'max_depth': 50, 'min_samples_split': 8, 'splitter': 'best'}
0.807 (+/-0.025) for {'criterion': 'entropy', 'max_depth': 50, 'min_samples_split': 8, 'splitter': 'random'}
0.807 (+/-0.037) for {'criterion': 'entropy', 'max_depth': 1000, 'min_samples_split': 6, 'splitter': 'best'}
0.807 (+/-0.052) for {'criterion': 'entropy', 'max_depth': 2000, 'min_samples_split': 5, 'splitter': 'random'}
0.807 (+/-0.035) for {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 5, 'splitter': 'best'}
0.807 (+/-0.037) for {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 7, 'splitter': 'best'}
0.807 (+/-0.042) for {'criterion': 'entropy', 'max_depth': 50, 'min_samples_split': 2, 'splitter': 'best'}
0.807 (+/-0.037) for {'criterion': 'entropy', 'max_depth': 200, 'min_samples_split': 6, 'splitter': 'best'}
0.807 (+/-0.032) for {'criterion': 'entropy', 'max_depth': 2000, 'min_samples_split': 8, 'splitter': 'random'}
0.808 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 20, 'min_samples_split': 5, 'splitter': 'best'}
0.808 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 100000, 'min_samples_split': 2, 'splitter': 'best'}
0.808 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 100000, 'min_samples_split': 4, 'splitter': 'random'}
0.808 (+/-0.044) for {'criterion': 'entropy', 'max_depth': 200, 'min_samples_split': 4, 'splitter': 'random'}
0.808 (+/-0.031) for {'criterion': 'entropy', 'max_depth': 5000, 'min_samples_split': 2, 'splitter': 'random'}
0.808 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 3, 'splitter': 'best'}
0.808 (+/-0.039) for {'criterion': 'entropy', 'max_depth': 1000, 'min_samples_split': 5, 'splitter': 'best'}
0.808 (+/-0.032) for {'criterion': 'entropy', 'max_depth': 5000, 'min_samples_split': 2, 'splitter': 'best'}
0.809 (+/-0.035) for {'criterion': 'entropy', 'max_depth': 200, 'min_samples_split': 7, 'splitter': 'best'}
0.809 (+/-0.039) for {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 3, 'splitter': 'random'}
0.810 (+/-0.033) for {'criterion': 'entropy', 'max_depth': 100, 'min_samples_split': 2, 'splitter': 'best'}
0.810 (+/-0.032) for {'criterion': 'entropy', 'max_depth': 2000, 'min_samples_split': 2, 'splitter': 'best'}
0.811 (+/-0.032) for {'criterion': 'entropy', 'max_depth': 5000, 'min_samples_split': 6, 'splitter': 'random'}
```

Najlepsze wyniki uzyskuje model z parametrem „splitter” z wartością „best”

Test no.3

Testowanie wartości parametru „max_depth”: od 5 do 100

```
0.685 (+/-0.075) for {'criterion': 'entropy', 'max_depth': 5, 'splitter': 'best'}
0.733 (+/-0.071) for {'criterion': 'entropy', 'max_depth': 6, 'splitter': 'best'}
0.769 (+/-0.069) for {'criterion': 'entropy', 'max_depth': 7, 'splitter': 'best'}
0.788 (+/-0.069) for {'criterion': 'entropy', 'max_depth': 8, 'splitter': 'best'}
0.809 (+/-0.057) for {'criterion': 'entropy', 'max_depth': 47, 'splitter': 'best'}
0.810 (+/-0.070) for {'criterion': 'entropy', 'max_depth': 9, 'splitter': 'best'}
0.811 (+/-0.065) for {'criterion': 'entropy', 'max_depth': 46, 'splitter': 'best'}
0.811 (+/-0.061) for {'criterion': 'entropy', 'max_depth': 89, 'splitter': 'best'}
0.811 (+/-0.059) for {'criterion': 'entropy', 'max_depth': 74, 'splitter': 'best'}
0.811 (+/-0.061) for {'criterion': 'entropy', 'max_depth': 52, 'splitter': 'best'}
0.812 (+/-0.057) for {'criterion': 'entropy', 'max_depth': 58, 'splitter': 'best'}
0.816 (+/-0.062) for {'criterion': 'entropy', 'max_depth': 98, 'splitter': 'best'}
0.816 (+/-0.058) for {'criterion': 'entropy', 'max_depth': 78, 'splitter': 'best'}
0.816 (+/-0.056) for {'criterion': 'entropy', 'max_depth': 87, 'splitter': 'best'}
0.816 (+/-0.053) for {'criterion': 'entropy', 'max_depth': 15, 'splitter': 'best'}
0.816 (+/-0.059) for {'criterion': 'entropy', 'max_depth': 25, 'splitter': 'best'}
0.816 (+/-0.067) for {'criterion': 'entropy', 'max_depth': 55, 'splitter': 'best'}
0.816 (+/-0.061) for {'criterion': 'entropy', 'max_depth': 70, 'splitter': 'best'}
0.816 (+/-0.058) for {'criterion': 'entropy', 'max_depth': 63, 'splitter': 'best'}
0.816 (+/-0.057) for {'criterion': 'entropy', 'max_depth': 69, 'splitter': 'best'}
0.817 (+/-0.055) for {'criterion': 'entropy', 'max_depth': 17, 'splitter': 'best'}
0.817 (+/-0.065) for {'criterion': 'entropy', 'max_depth': 34, 'splitter': 'best'}
0.818 (+/-0.059) for {'criterion': 'entropy', 'max_depth': 92, 'splitter': 'best'}
0.818 (+/-0.057) for {'criterion': 'entropy', 'max_depth': 86, 'splitter': 'best'}
0.818 (+/-0.057) for {'criterion': 'entropy', 'max_depth': 11, 'splitter': 'best'}
0.818 (+/-0.056) for {'criterion': 'entropy', 'max_depth': 67, 'splitter': 'best'}
0.819 (+/-0.059) for {'criterion': 'entropy', 'max_depth': 45, 'splitter': 'best'}
```

Wartości parametru „max_depth” poniżej 10 coraz bardziej obniżają jakość modelu.

W przedziale wartości od 10 do 100 dla parametru „max_depth” nie obserwujemy większej zależności.

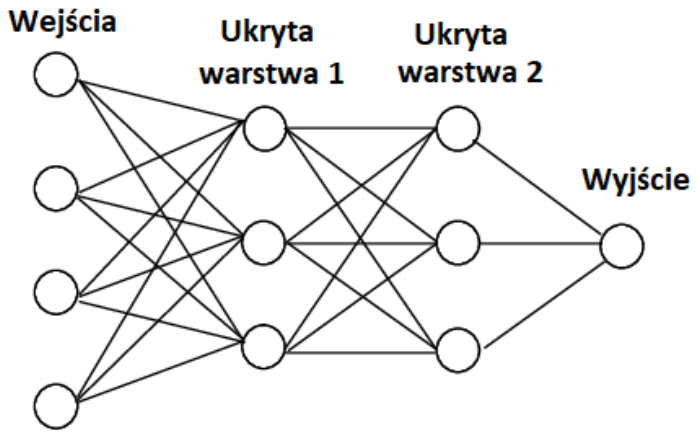
„criterion” – Funkcja do pomiaru jakości podziału.

„max_depth” – maksymalna głębokość drzewa.

„splitter” – Strategia użyta do wyboru podziału w każdym węźle.

Sieć neuronowa czyli Multi-Layer-Perceptron

MLP składa się z co najmniej trzech warstw węzłów: warstwy wejściowej, warstwy ukrytej i warstwy wyjściowej. Z wyjątkiem węzłow wejściowych, każdy węzeł jest neuronem wykorzystujący nieliniową funkcję aktywacji. MLP wykorzystuje technikę nadzorowanego uczenia się zwaną wsteczną propagacją do treningu.



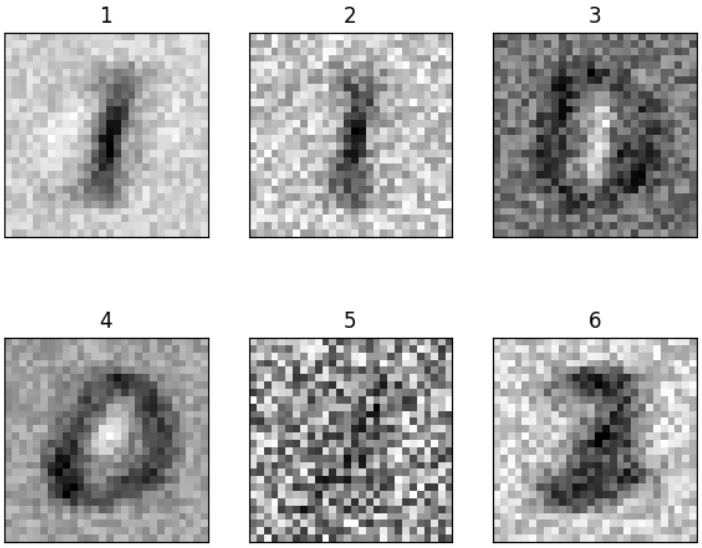
Wartości na węzłach na poprzedniej warstwie są mnożone przez współczynniki na połączeniach a następnie sumowanie jako wartość węzła. Ten proces jest powtarzany aż w końcu otrzymamy wynik. Gdy zadaniem naszego modelu jest klasyfikacja, wyjść może być wiele i zazwyczaj wartości na węzłach odpowiadają „pewności" że taki jest wynik. W naszym wypadku model może podejrzewać z dużą pewnością że jakaś liczba to 8 oraz 0. Wybierany jest najpewniejszy wynik. Niestety jednak komputer „nie widzi" liczb tak jak my i dla dużej ilości węzłów oraz warstw trudno zrozumieć działanie poszczególnych węzłów.

Zagadnienie sieci neuronowych oraz algorytmów używanych do ich działania oraz trenowania mogły by na osobności stanowić temat obszernego projektu. Z racji na ograniczony procent treści merytorycznej dokładniejsze objaśnienie mija się z celem.

Sprawdźmy to za pomocą wartości współczynników dla atrybutów dla każdego węzła. W ten sposób możemy zobaczyć co najbardziej wpływa na wartość naszego węzła.

Poniżej przedstawione są wyniki dla modelu o 6 warstwach, uwzględniającego 4 klasy czyli cyfry od 0 do 3.

Oto 6 wykresów przedstawiających 6 węzłów. Obraz który widzimy to współczynniki dla każdego atrybutu('piksela'). To nie są liczby ze zbioru MINST, człowiek tych cyfr nie napisał. To co widzimy to percepcja modelu.



No dobrze więc gdzie tutaj jest 3? No nie ma. Według naszego spojrzenia nie ma jej tutaj. Ale to nie my decydujemy, tylko komputer.

Obejrzyjmy wartości współczynników dla wyjść. Mamy 4 cyfry czyli 4 wyjścia. Wypisujemy współczynniki powiązania węzłów ukrytej warstwy z wyjściami.

```
Number : 0                          Number : 2
-0.4699893641182047 for node nr 4   -0.670176310464558 for node nr 2
-0.4214056164471968 for node nr 3   -0.3646723930001858 for node nr 6
-0.12451411754229766 for node nr 6  -0.15255235878053894 for node nr 5
0.3191641299592913 for node nr 5    0.048181342235560885 for node nr 3
0.37881635533813846 for node nr 2   0.29894499432567706 for node nr 1
0.4058216847803588 for node nr 1    1.1336337800562872 for node nr 4
Number : 1                          Number : 3
-0.6859264831613191 for node nr 2   -1.850286843710424 for node nr 4
-0.6099623195986112 for node nr 5   -0.6383505623268286 for node nr 5
-0.3856857419284567 for node nr 1   -0.5927844795095832 for node nr 2
-0.2780039589976377 for node nr 6   -0.10660933844088956 for node nr 3
0.09093672235162166 for node nr 3   -0.06379306792146983 for node nr 6
1.5577489518559795 for node nr 4    0.08678171307655816 for node nr 1
```

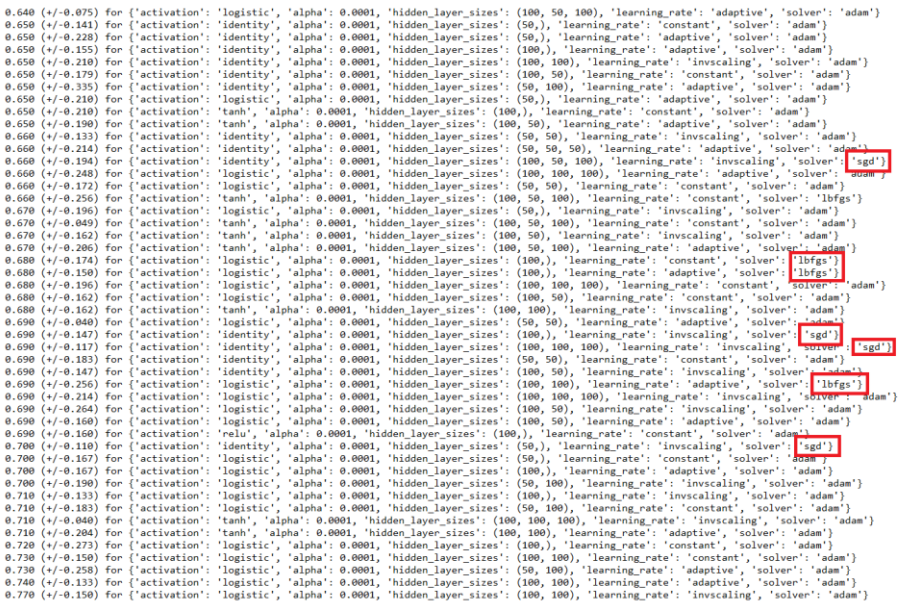
Z tych danych wyczytujemy że za uzyskanie odpowiedzi: 1 najbardziej odpowiadają węzły 2,5,1 , co ma sens, na obrazie węzła 5 trudno spostrzec cokolwiek, ale na 1 i 2 widzimy jedynkę.

Za zero odpowiada natomiast 4,3 i widzimy tam okrągły kształt.

Test no.1

Testowane wartości parametru „solver”:

- Adam
- Sgd
- Lbfgs



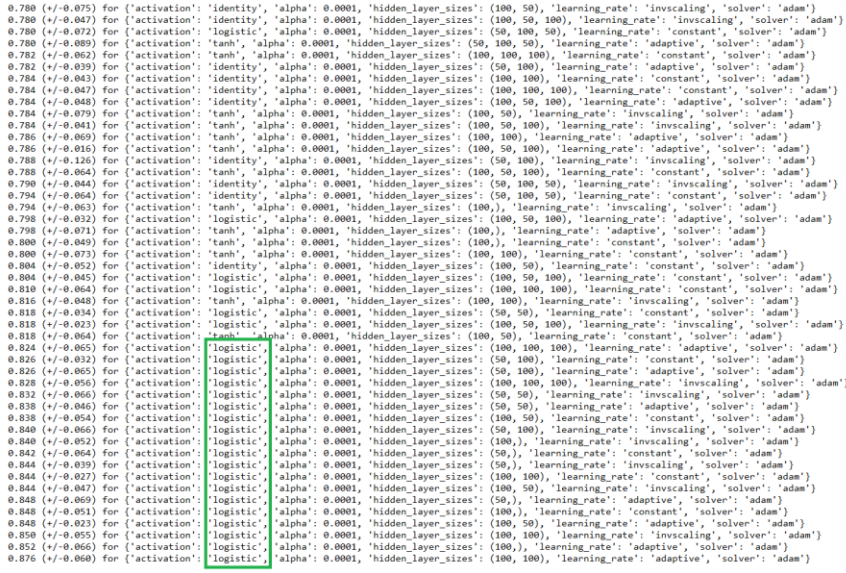
Inne wartości dla parametru „solver” niż „adam” nie generują najlepszych modeli.

(„solver” z wartością „adam” nie wymaga doprecyzowania parametru „learning_rate”)

Test no.2

Testowanie wartości parametru „activation”:

- Tanh
- Identity
- Logistic
- Relu



W przedziale wartości od 10 do 100 dla parametru „max_depth” nie obserwujemy większej zależności.

Test no.3

```
0.861 (+/-0.046) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'adam'}
0.864 (+/-0.030) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'invscaling', 'solver': 'adam'}
0.866 (+/-0.041) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'invscaling', 'solver': 'adam'}
0.868 (+/-0.049) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'adam'}
0.872 (+/-0.051) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.877 (+/-0.049) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.883 (+/-0.063) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'invscaling', 'solver': 'adam'}
0.886 (+/-0.062) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50, 100), 'learning_rate': 'constant', 'solver': 'adam'}
0.887 (+/-0.038) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant', 'solver': 'adam'}
0.887 (+/-0.030) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 100, 100), 'learning_rate': 'constant', 'solver': 'adam'}
0.888 (+/-0.037) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.889 (+/-0.054) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100), 'learning_rate': 'constant', 'solver': 'adam'}
0.889 (+/-0.049) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.889 (+/-0.051) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 100, 100), 'learning_rate': 'invscaling', 'solver': 'adam'}
0.889 (+/-0.046) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100), 'learning_rate': 'invscaling', 'solver': 'adam'}
0.891 (+/-0.048) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'learning_rate': 'constant', 'solver': 'adam'}
0.891 (+/-0.037) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50, 100), 'learning_rate': 'invscaling', 'solver': 'adam'}
0.892 (+/-0.051) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'learning_rate': 'invscaling', 'solver': 'adam'}
0.893 (+/-0.036) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50, ), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.893 (+/-0.047) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50, 100), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.897 (+/-0.039) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, ), 'learning_rate': 'invscaling', 'solver': 'adam'}
0.899 (+/-0.041) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 100), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.899 (+/-0.028) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 100, 100), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.901 (+/-0.042) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, ), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.902 (+/-0.040) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, ), 'learning_rate': 'constant', 'solver': 'adam'}
0.902 (+/-0.051) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 100), 'learning_rate': 'constant', 'solver': 'adam'}
0.903 (+/-0.060) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 100), 'learning_rate': 'invscaling', 'solver': 'adam'}
0.903 (+/-0.042) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.905 (+/-0.041) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50), 'learning_rate': 'constant', 'solver': 'adam'}
0.905 (+/-0.038) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50), 'learning_rate': 'invscaling', 'solver': 'adam'}
```

Pierwsza ukryta warstwa powinna zawierać 100 neuronów.

Test no.4

```
0.918 (+/-0.025) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50, 100), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.921 (+/-0.020) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 100, 100), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.924 (+/-0.020) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 100), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.929 (+/-0.023) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.936 (+/-0.022) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'adam'}
```

Najlepszy model jest dla jednej ukrytej warstwy o wartości 100

Linear Regression

Miara korelacji (Pearsona) pozwala na stwierdzenie stopnia zależności liniowej atrybutów (cech).

Test no.1

Testujemy wartości parametru „positive”: 0 lub 1

```
-732887970861286185869770752000.000 (+/-43076244710394763784332981043 2.000) for {'copy_X': 0, 'fit_intercept': 1, 'n_jobs': 4, 'normalize': 1, 'positive': 0}
-732887970861286185869770752000.000 (+/-43076244710394763784332981043 2.000) for {'copy_X': 1, 'fit_intercept': 1, 'n_jobs': 4, 'normalize': 1, 'positive': 0}
-455140919665176412 16.000 (+/-2424640377710045429 76.000) for {'copy_X': 0, 'fit_intercept': 1, 'n_jobs': 4, 'normalize': 0, 'positive': 0}
-455140919665176412 16.000 (+/-2424640377710045429 76.000) for {'copy_X': 1, 'fit_intercept': 1, 'n_jobs': 4, 'normalize': 0, 'positive': 0}
-175710588739531 3.250 (+/-1045421187031504 0.000) for {'copy_X': 0, 'fit_intercept': 0, 'n_jobs': 4, 'normalize': 0, 'positive': 0}
-175710588739531 3.250 (+/-1045421187031504 0.000) for {'copy_X': 0, 'fit_intercept': 0, 'n_jobs': 4, 'normalize': 1, 'positive': 0}
-175710588739531 3.250 (+/-1045421187031504 0.000) for {'copy_X': 1, 'fit_intercept': 0, 'n_jobs': 4, 'normalize': 0, 'positive': 0}
-175710588739531 3.250 (+/-1045421187031504 0.000) for {'copy_X': 1, 'fit_intercept': 0, 'n_jobs': 4, 'normalize': 1, 'positive': 0}
0.476 (+/-0.111) for {'copy_X': 0, 'fit_intercept': 0, 'n_jobs': 4, 'normalize': 0, 'positive': 1}
0.476 (+/-0.111) for {'copy_X': 0, 'fit_intercept': 0, 'n_jobs': 4, 'normalize': 1, 'positive': 1}
0.476 (+/-0.111) for {'copy_X': 1, 'fit_intercept': 0, 'n_jobs': 4, 'normalize': 0, 'positive': 1}
0.476 (+/-0.111) for {'copy_X': 1, 'fit_intercept': 0, 'n_jobs': 4, 'normalize': 1, 'positive': 1}
0.484 (+/-0.102) for {'copy_X': 0, 'fit_intercept': 1, 'n_jobs': 4, 'normalize': 0, 'positive': 1}
0.484 (+/-0.102) for {'copy_X': 1, 'fit_intercept': 1, 'n_jobs': 4, 'normalize': 0, 'positive': 1}
0.484 (+/-0.102) for {'copy_X': 0, 'fit_intercept': 1, 'n_jobs': 4, 'normalize': 1, 'positive': 1}
0.484 (+/-0.102) for {'copy_X': 1, 'fit_intercept': 1, 'n_jobs': 4, 'normalize': 1, 'positive': 1}
```

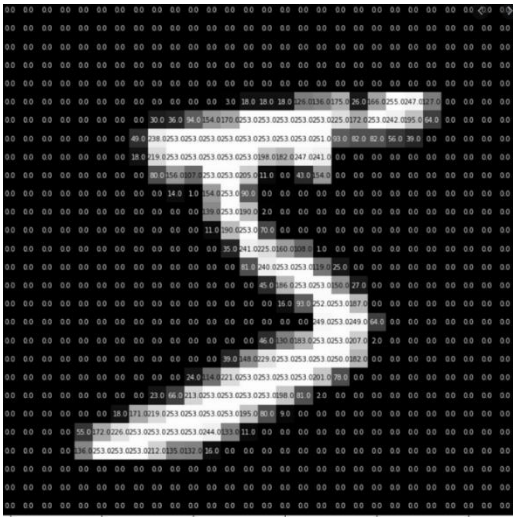
Rozwiązanie problemu metodą regresji liniowej jest wyjątkowo nieefektowne

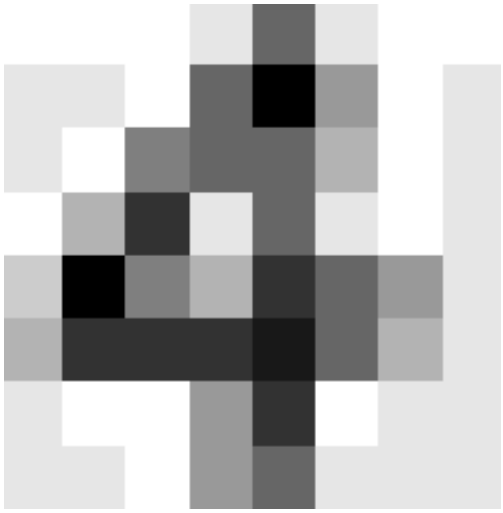
Widzimy, że wartość parametru „positive” musi być ustawiona na 1, żeby model w ogóle działał.

Żeby rozwiązywanie problemu metodą liniową miało sens powinniśmy mocno zmienić dane wejściowe.

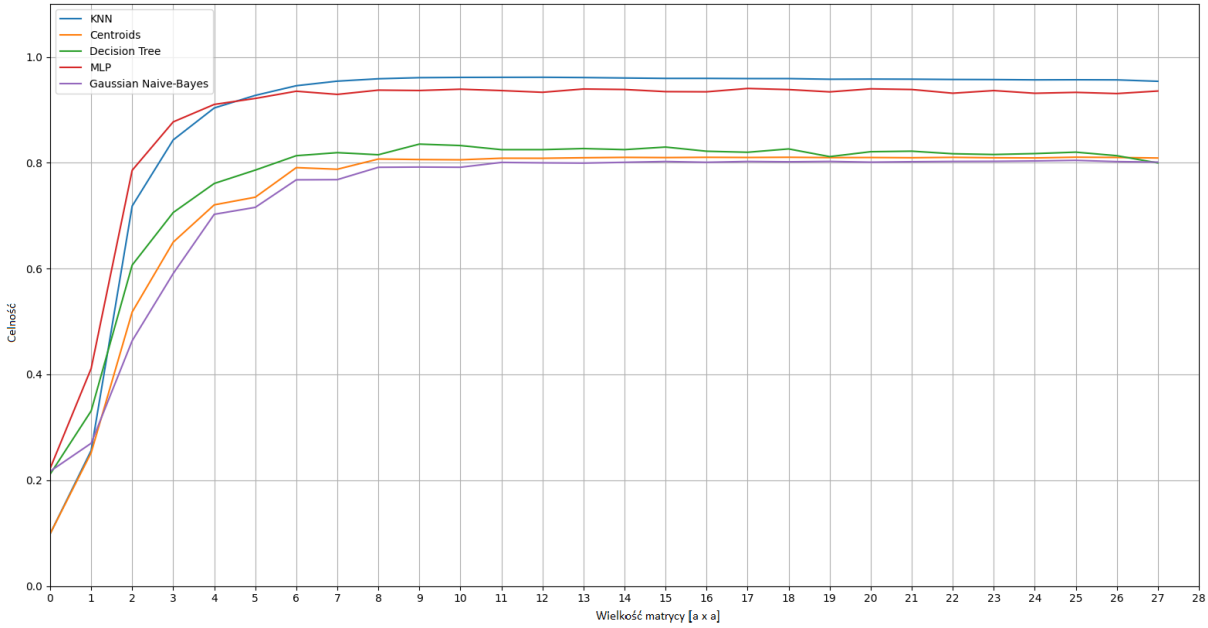
3. Analiza uzyskanych wyników

W celu minimalizacji czasu jaki metoda będzie potrzebowała do wyprodukowania wyniku na oryginalnych danych można przeprowadzić kompresję.



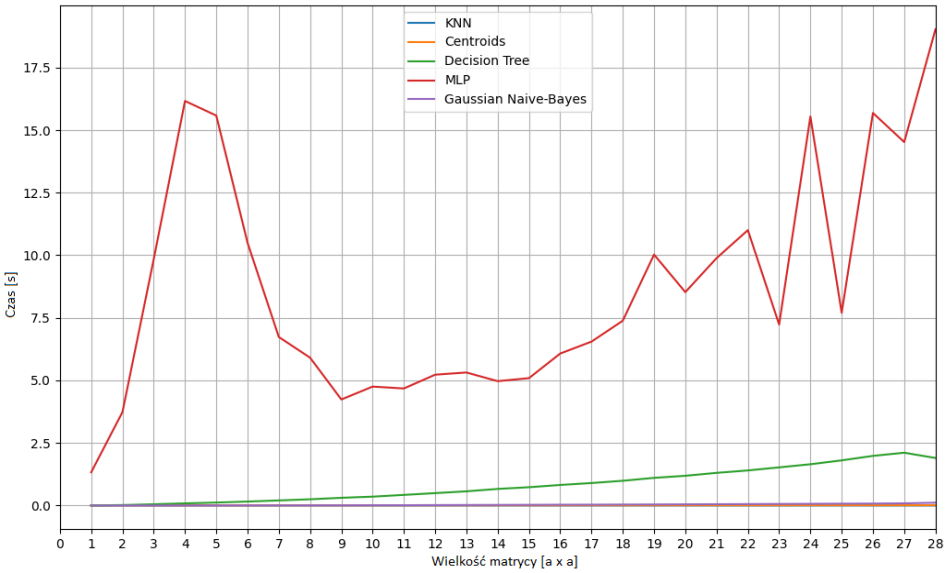


Początkowo zdjęcie jest matrycą 28*28 pikseli w skali szarości. Więc łatwo się skaluje, bo mamy tylko jedną wartość w pikselu z których później liczymy średnią arytmetyczną i znajdujemy nowe mniejsze piksele.

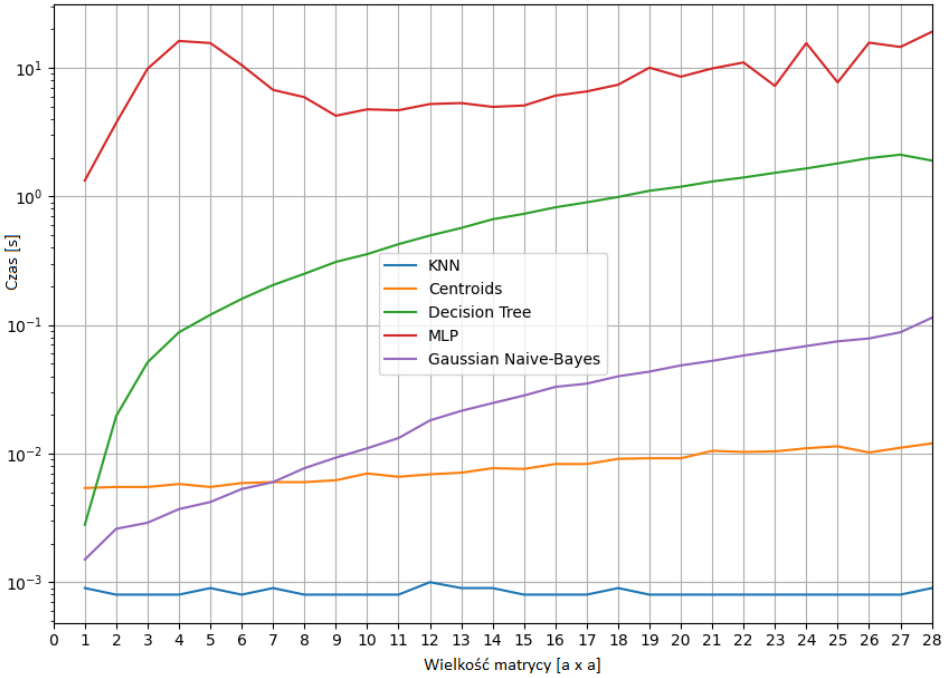


Z tego wykresu widzimy, że większość metod poprawnie działa gdy mają co najmniej 8*8 pikseli.

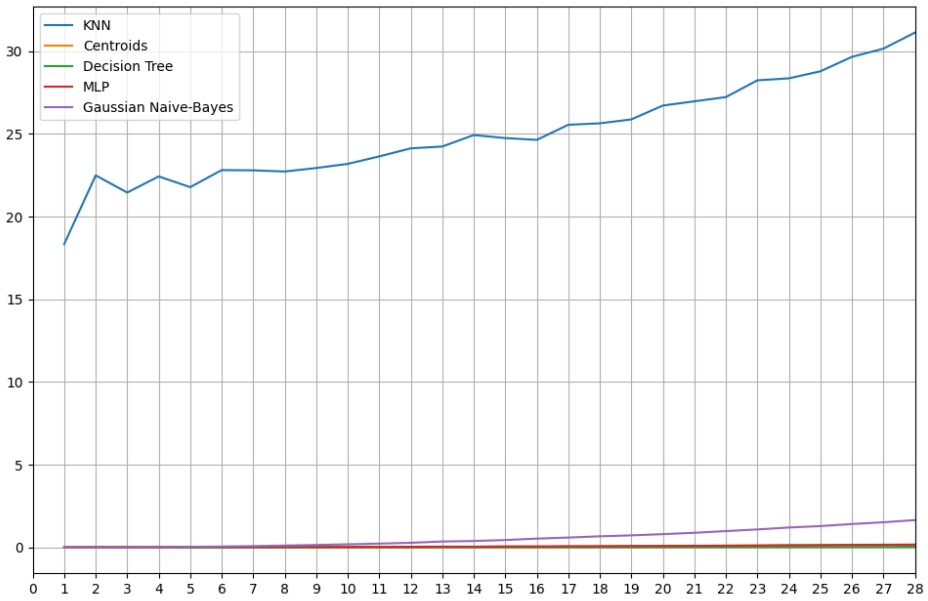
Widzimy także, że najlepiej spisuje się metoda k – średnich sąsiadów. Jednak w przedziale, gdy mamy małą matryce MLP radzi sobie znacznie lepiej. Skuteczność tych metod jest powyżej 93%. Istnieje także druga grupa metod złożona z Decision Tree, Gaussa i Cetrodów, których skuteczność oscyluje wokół 80%.



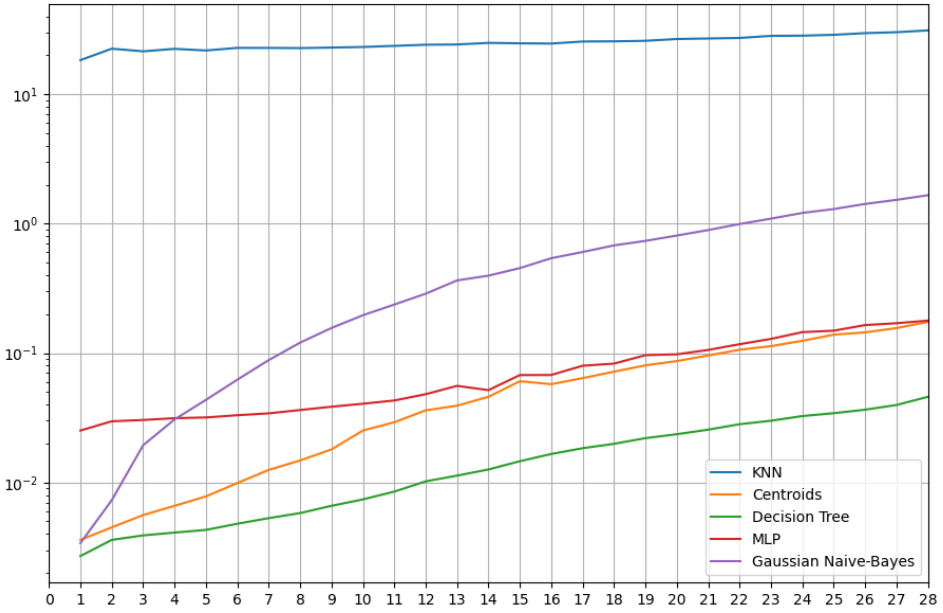
Na pierwszym wykresie dobrze widać, że MLP potrzebuje dużo czasu i nie koniecznie wprost proporcjonalnego do ilości danych.
Także widzimy, że drzewo decyzyjne lekko wzrasta sugerując złożoność super liniową.



Dzieję się tak, ponieważ zarówno MLP, jak i drzewo decyzyjne, wymagają trenowania na danych. Pozostałe modele tego treningu nie potrzebują.



Z kolei na tym wykresie obserwujemy, że modele które nie polegają na treningu mają znacznie dłuższe czasy klasyfikacji. Ta zależność jest jednym z powodów, dlaczego istnieje tak wiele modeli, a także tak wiele z nich nadal jest używanych.



Drugi wykres znacznie lepiej obrazuje różnice najlepszych modeli. Widzimy że drzewo decyzyjne działa jak zakładano, po czasie około liniowym. Metoda Gausowska natomiast bardzo szybko powoduje duże opóźnienia.

Moglibyśmy wyobrazić sobie sytuację, w której potrzebny byłby model potrafiący zgadywać niemal natychmiastowo po otrzymaniu danych wstępnych. Z kolei na drugiej stronie spektrum widzimy modele, które utworzone raz błyskawicznie przewidywałyby nowe wpływające dane.

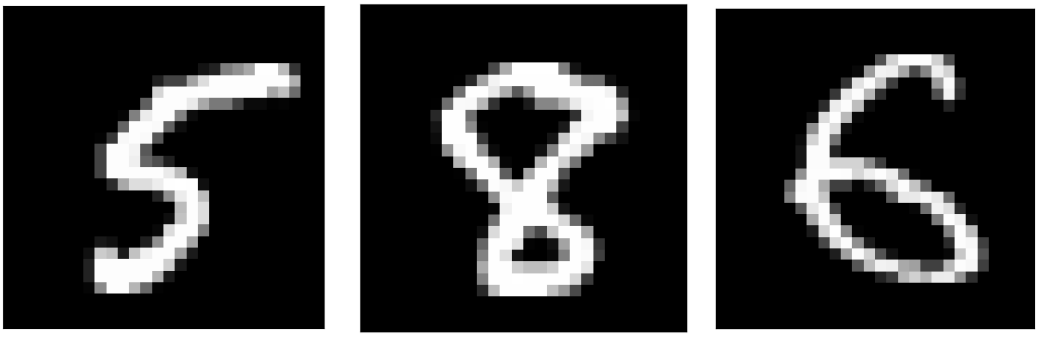
Najlepszy model

Na podstawie naszych wyników widzimy, że najlepszą celność posiada model KNN (K-nearest neighbors) z parametrami n_neighbors = 4, weights = 'distance', algorithm = 'brute', metric = 'cosine'. Ponieważ model ten nie wymaga prawie żadnego trenowania, to możemy go niemal natychmiastowo utworzyć. Oczywiście możemy spodziewać się, że modele takie jak drzewo decyzyjne będą o wiele szybciej dokonywać klasyfikacji.

4. Używanie modeli do przywidywania cyfr

Udane próby klasyfikacji

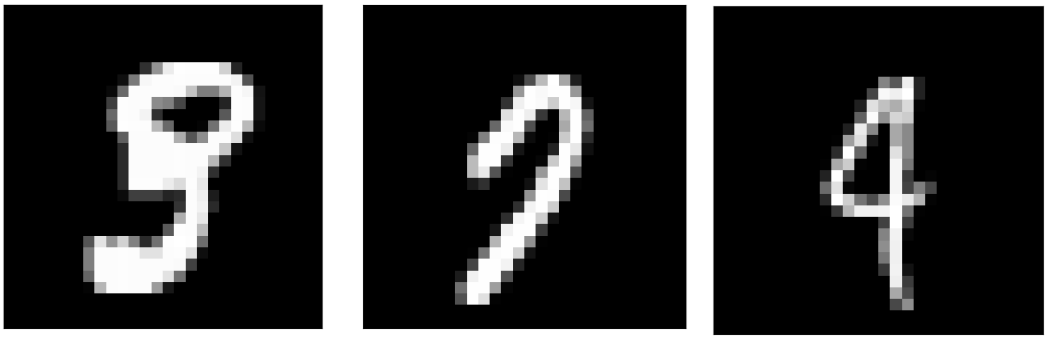
Poniżej są przedstawione udane klasyfikacje przy użyciu wybranego modelu.



Myślę, że wyniki są naprawdę niezwykłe. Głupia maszyna zdaję się przejawiać zrozumienie, inteligencję. Tym bardziej, że większość z tych testów osądzana jest szybciej, niż mógłby to wykonać człowiek.

Nieudana próba klasyfikacji

Zobaczmy teraz kiedy program popełnia błędy. Poniżej znajdują się 3 przykłady niepoprawnie oszacowanych cyfr.



Pierwsza cyfra została przewidziana jako 8, jednak według autora jest to 3. Widzimy, że górny ogonek trójki złączył się ze środkowym. Gdyby jeszcze połączyć środkowy z dolnym, to wyszłaby

dość wyraźna ósemka. Mimo że widzimy tam przerwę trzeba pamiętać, że żaden z modeli nie polega na kształcie jako element decyzyjny. W większości wypadków jest to pokrycie otrzymanego obrazu z idealnym wzorcem danej liczby.

Drugi obrazek ma zestaw przewidziany jako 9, natomiast oryginalne jest to 7. Przeciętny człowiek również z pewnością miałby problemy z określeniem tego przypadku. Jednak nie byłby to spor pomiędzy 7 i 9 a miedzy 7 i 1. Ta różnica jest spowodowana innym sposobem określania cyfry, człowiek patrzy na kształt, a nasz program na jasność konkretnych pikseli.

Tutaj wyjaśnia się absurdalny przypadek cyfry 4 podobnej. Program na tym obrazku „zobaczył" 9, co jest logiczne biorąc pod uwagę, że zmiana jasności pikseli jest wręcz identyczna do zmian jasności występujących u typowych przypadków cyfry 9. Jedyna różnica to mały koniuszek lekko występujący poza linie poziomą. Dla ludzkiego oka to jest oczywisty sygnał, a dla programu mały szczegół.