

Sprawozdanie

Algorytmy Uczenia Maszynowego – Projekt

1. Opis problemu

Głównym założeniem projektu było stworzenie programu, który za pomocą wybranego z zaimplementowanych algorytmów uczenia maszynowego ocenić, czy podany artykuł nadaje się do publikacji. Po konsultacji z prowadzącymi zmieniono temat projektu na klasyfikację, czy podany krótki tekst (tweet) nadaje się do publikacji.

Zmiana ta miała miejsce, ponieważ były problemy z stworzeniem odpowiednio dużej bazy artykułów w języku polskim do „nauczenia” modelu.

2. Użyte algorytmy

- **SVC** – Support Vector Classification

Jest to algorytm, którego czas działania funkcji **fit** wzrasta co najmniej kwadratowo wraz z wzrostem ilości próbek, co powoduje, że jest on niepraktyczny dla dużych zbiorów danych. W dokumentacji możemy znaleźć informację od developerów, że dla większych zbiorów danych polecają algorytmy „**LinearSVC**” oraz „**SGDClassifier**” po transformacji za pomocą funkcji „**Nystroem**”.

Przykładowe parametry, którymi można zmienić działanie algorytmu:

- **kernel** – Za pomocą tego parametru możemy zmienić typ jądra, za pomocą którego algorytm pracuje. Dostępne opcje to: **linear**, **poly**, **rbf**, **sigmoid**, **precomputed**. Domyślnie algorytm używa jądra **rbf**.
- **C** – Stopień regulacji. Siła regulacji jest odwrotnie proporcjonalna do wartości zmiennej **C** i zawsze musi przyjmować wartości dodatnie.
- **tol** – Współczynnik tolerancji dla kryterium zatrzymania. Domyślna wartość wynosi 10^{-3} .
- **Max_iter** - Współczynnik określający maksymalną liczbę iteracji. Domyślna wartość wynosi **-1**, czyli bez limitu.
- **Class_weight** – Parametr nadający wagę poszczególnym klasom. Domyślnie algorytm przypisuje wszystkim klasom tą samą wagę równą **1**. Po podaniu wartości „**balanced**” algorytm automatycznie będzie dobierał wagi odwrotnie proporcjonalnie do ilości wystąpień danej klasy w zbiorze danych wejściowych.

- **MLP** - Multi-layer Perceptron

Jest to algorytm sieci neuronowej, który optymalizuje błąd kwadratowy predykcji wykorzystując **algorytm optymalizacyjny LBFGS** lub **stochastyczny gradient spadku**.

Zalety MLP zaprezentowane w dokumentacji:

- Możliwość nauki modeli nieliniowych
- Możliwość uczenia modeli w czasie rzeczywistym za pomocą „**partial_fit**”.

Wady MLP wypunktowane w dokumentacji:

- MLP ma niewypukłą funkcję strat, gdzie istnieje więcej niż jedno minimum lokalne. Dodatkowo, inicjalizacje o różnych, losowych wagach mogą prowadzić do innych dokładności walidacji.
- MLP wymaga określenia „hiperparametrów” takich jak liczba ukrytych neuronów, warstw i iteracji.
- MLP jest wrażliwe na skalowanie funkcji.

Przykładowe parametry, którymi można zmienić działanie algorytmu:

- **solver** – za pomocą tego parametru wybieramy funkcję optymalizującą. Możliwe wartości to **lbfgs**, **sgd** oraz **adam**. Dwie pierwsze wymienione wartości odnoszą się do metod optymalizacyjnych wymienionych wyżej, natomiast wartość „**adam**” odnosi się do optymalizatora opartego na stochastycznych gradientach zaproponowanym przez Kingmę, Diederik oraz Jimmy Ba. Domyślną wartością dla parametru **solver** jest wartość „**adam**”.
- **learning_rate** – za pomocą tego parametru możemy określić, które dane wejściowe będą miały większy wpływ na naukę naszego modelu. Możliwe wartości to:
 - **constant** – wszystkie dane mają taką samą wagę,
 - **invscaling** - waga kolejnych podawanych danych spada.
 - **adaptive** - waga kolejnych danych jest taka sama jak wartość inicjalizacyjna, o ile strata treningowa maleje. Za każdym razem, gdy 2 kolejne epoki nie powodują spadku błędu lub nie powodują wzrostu współczynnika walidacji, to obecna waga danych uczonych jest dzielona przez 5.
- **tol** – Współczynnik tolerancji dla optymalizacji. Domyślna wartość wynosi 10^{-4} . Jeśli „**learning_rate**” nie jest ustawiony na „**adaptive**” oraz strata lub wynik nie zmienia się o co najmniej wartość „**tol**” to algorytm kończy pracować.

- **K Najbliższych sąsiadów**

Jeden z algorytmów regresji nieparametrycznej używany w statystyce do prognozy wartości pewnej zmiennej losowej. Możliwe jest użycie go w zadaniach klasyfikacji.

Przykładowe parametry, którymi można zmienić działanie algorytmu:

- **n_neighbors** - kluczowy parametr w algorytmie. Określa liczbę sąsiadów, z którymi będą porównywane wyniki.
- **weights** – parametr określający wagi sąsiadów. Możliwe są następujące wartości:
 - **uniform** - każdy punkt w sąsiedztwie ma taką samą wagę. Jest to domyślna wartość dla algorytmu.
 - **distance** – wraz z wzrostem odległości spada waga danego punktu.
 - **callable** - jest to funkcja określona przez użytkownika, która musi być przyjmować wektor odległości i zwracać wektor wag, przy czym oba wektory muszą mieć ten sam rozmiar.
- **algorithm** – parametr określający, który algorytm będzie używany do obliczania najbliższych sąsiadów. Możliwe są następujące wartości:
 - **ball_tree** – używa funkcji BallTree. W bibliografii umieszczono link do dokumentacji tej funkcji.
 - **kd_tree** – używa funkcji KDTree. W bibliografii umieszczono link do dokumentacji tej funkcji.
 - **brute** – używa metody brute-force przeglądu
 - **auto** – algorytm spróbuje sam dobrać najlepszy algorytm z wyżej wymienionych na podstawie danych wejściowych podanych w funkcji **fit**.

3. Podstawowe miary jakości

W programie zaimplementowano następujące miary jakości:

- **accuracy_score** – Zgodnie z dokumentacją, współczynnik ten wylicza dokładność podzbiorów, gdzie zestaw przewidywanych etykiet musi dokładnie odpowiadać odpowiadającemu zestawowi rzeczywistych etykiet.
- **precision_score** – Współczynnik wskazujący iloraz poprawnych klasyfikacji oraz wszystkich klasyfikacji. Wartości znajdują się w przedziale [0,1], gdzie **1** to najlepsza wartość, a **0** to najgorsza wartość.
- **f1_score** – Współczynnik ten można uznać za średnią harmoniczną. Podobnie jak współczynnik **precision_score**, przyjmuje wartości z przedziału [0,1], gdzie **1** to najlepsza wartość, a **0** to najgorsza wartość. Wartość współczynnika **f1_score** jest wyliczana z następującego wzoru:

$$f1_score = \frac{2*precision*recall}{precision+recall},$$

gdzie precision jest przedstawione wzorem:

$$precision = \frac{True_positive}{True_positive + false_positive}$$

oraz recall jest przedstawione wzorem:

$$recall = \frac{True_positive}{True_positive + false_negatives}$$

- **confusion_matrix** – współczynnik ten wskazuje nam, ile danych błędnie sklasyfikowano. Można to zobrazować w następujący sposób:

True negatives	False positives
False negatives	True positives

gdzie „**True negatives**” to **0** sklasyfikowane jako **0**, „**False negatives**” to **1** sklasyfikowane jako **0**, „**False positives**” to **0** sklasyfikowane jako **1** a „**True positives**” to **1** sklasyfikowane jako **1**.

4. Porównanie wyników algorytmów dla tych samych danych testowych

Algorytm SVC:

```
Accuracy: 0.5666666666666667
Precision: [0.5 1. ]
F1 score: [0.66666667 0.38095238]
Confusion matrix: [[13  0]
 [13  4]]
Working time in nanoseconds: 0
Koniec statystyk.
```

Rysunek 1 - Statystyki algorytmu SVC

Algorytm KNN dla n=5:

```
Accuracy: 0.43333333333333335
Precision: [0.43333333 0.        ]
F1 score: [0.60465116 0.        ]
Confusion matrix: [[13  0]
 [17  0]]
Working time in nanoseconds: 1958300
Koniec statystyk.
```

Rysunek 3 - Statystyki algorytmu KNN

Algorytm MLP dla 100 iteracji:

```
Accuracy: 0.6333333333333333
Precision: [0.55555556 0.75        ]
F1 score: [0.64516129 0.62068966]
Confusion matrix: [[10  3]
 [ 8  9]]
Working time in nanoseconds: 997800
Koniec statystyk.
```

Rysunek 2 - Statystyki algorytmu MLP

Wszystkie trzy algorytmy zostały przetrenowane na tym samym zestawie danych uczących, a statystyki zostały wyliczone na takich samych danych testowych o liczebności 30 rekordów. Można zauważyć, że algorytm MLP osiąga najlepszą dokładność, chociaż ta nadal nie jest wystarczająco wysoka, aby można jej było praktycznie używać. Jest to spowodowane zbyt małą bazą danych o niezbyt dużej różnorodności danych zawartej w bazie. Gdyby baza ucząca została lepiej przygotowana, to dokładność wszystkich 3 zaimplementowanych algorytmów wzrosłaby.

Ze wszystkich zaimplementowanych algorytmów najszybciej pracuje SVC. Drugim pod względem prędkości predykcji jest algorytm MLP, który także osiągał najwyższą dokładność predykcji. Najwolniej, a także najgorsze wyniki osiągał algorytm K Najbliższych Sąsiadów.

5. Połączenie algorytmów

Aby usprawnić działanie algorytmów i spróbować zmniejszyć wpływ błędów poszczególnych metod, została utworzona funkcja „UsageCombined”, w której każda funkcja wykonuje swoje predykcje odnośnie podanych danych, a następnie wyliczana jest średnia ważona z wyliczonych predykcji. Waga predykcji danego algorytmu jest tym wyższa, im większa jest dokładność tegoż algorytmu.

```
clf1 = pickle.load(open("model_SVC.sav", 'rb'))
clf2 = pickle.load(open("model_MLP.sav", 'rb'))
clf3 = pickle.load(open("model_KNN.sav", 'rb'))
Y1_predicted = clf1.predict(X_to_predict)
Y2_predicted = clf2.predict(X_to_predict)
Y3_predicted = clf3.predict(X_to_predict)
Y_predicted = Y2_predicted*0.4 + Y1_predicted*0.35 + Y3_predicted*0.25
Y_predicted = [int(val + 0.5) for val in Y_predicted]
```

Rysunek 4 - Fragment programu pokazujący połączenie algorytmów.

6. Praktyczne używanie programu

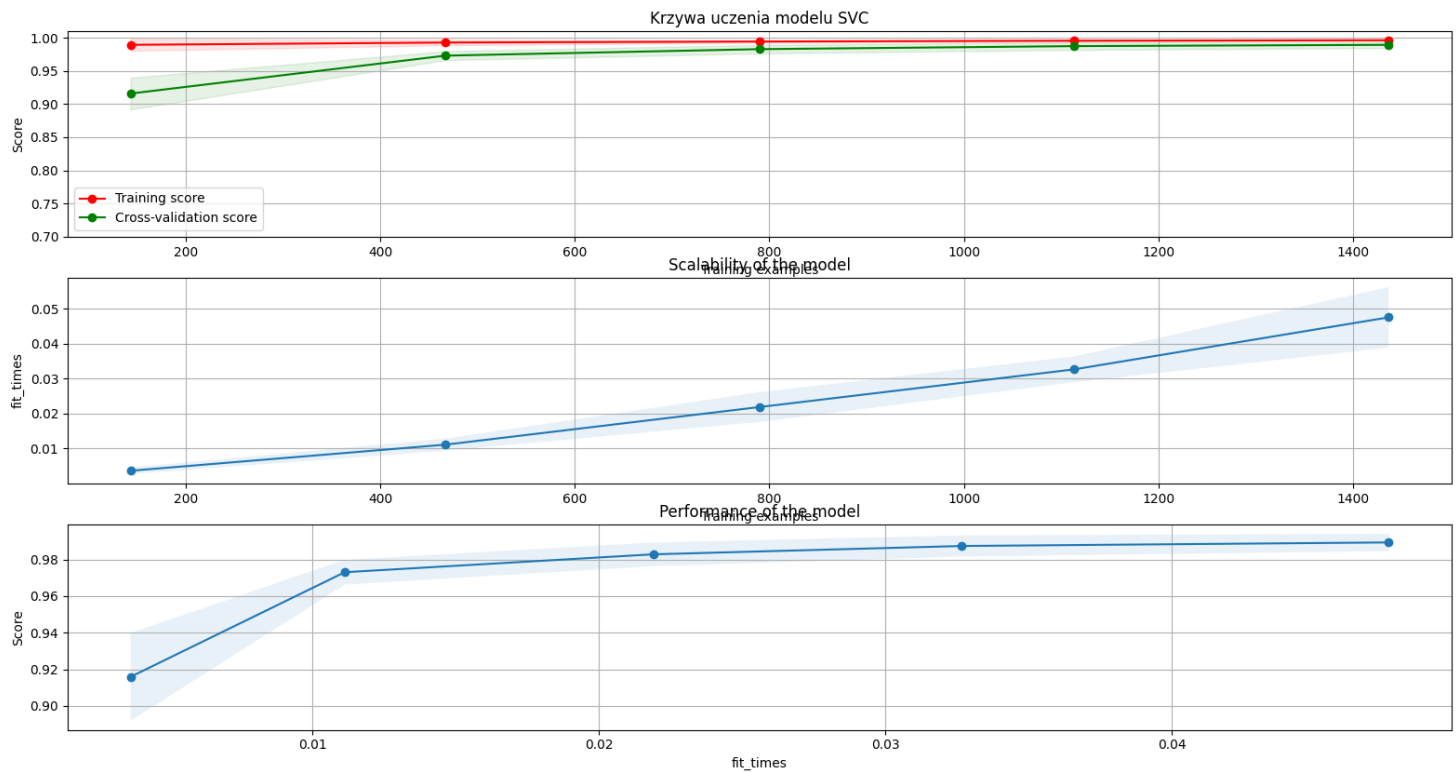
Aby program poprawnie działał, na początku należy wybrać algorytm uczenia maszynowego, który ma być wykorzystany do predykcji. Następnie należy podać plik z danymi wejściowymi – najlepiej w rozszerzeniu „.txt”. Następnie program dokona predykcji wykorzystując stworzony model, po czym wypisze swoje predykcje na każdy rekord znajdujący się w pliku wejściowym.

```
Algorytm: MLP
Wyniki: [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 1]
```

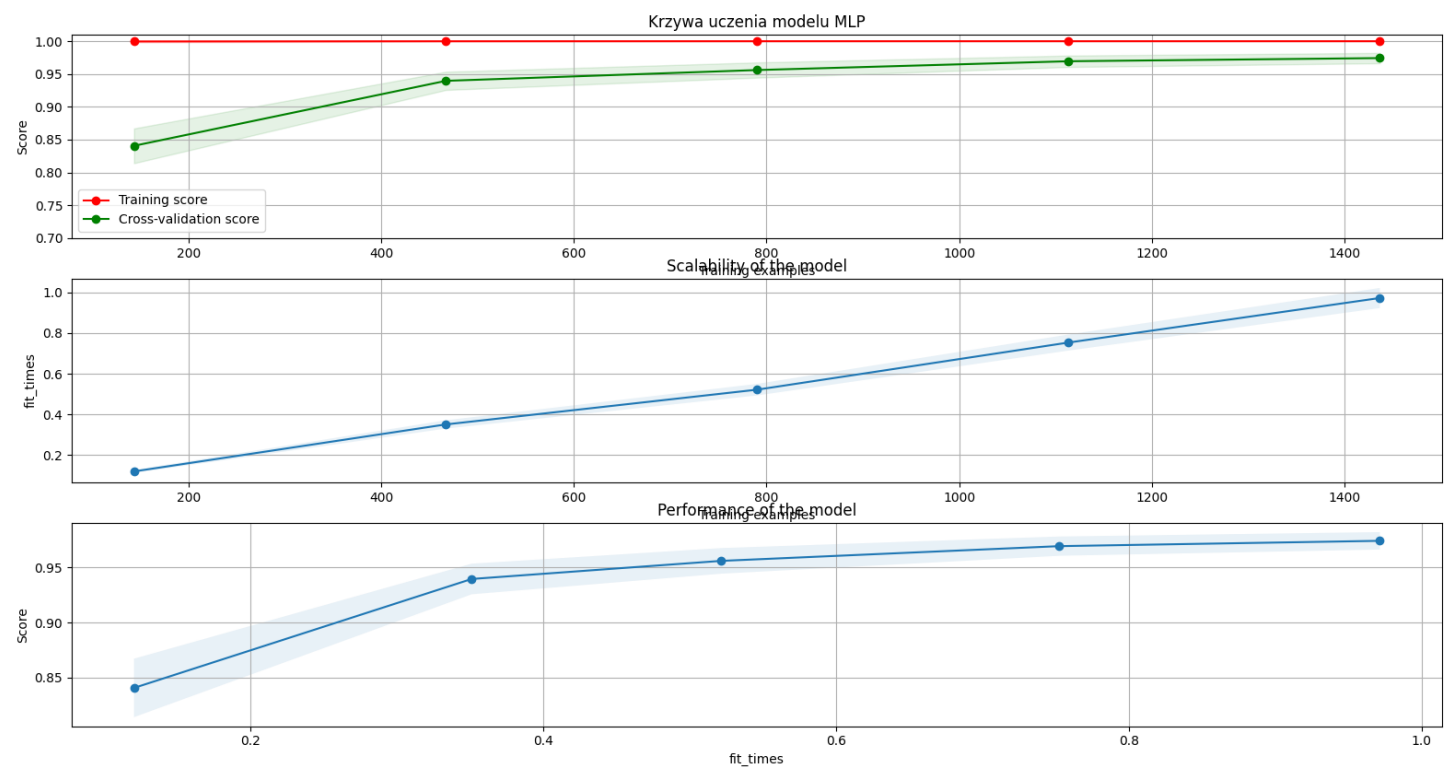
Rysunek 5 - Wynik działania algorytmu dla podanych danych testowych.

7. Wykresy

Na wykresach poniżej przedstawiono krzywe uczenia modeli SVC oraz MLP.



Rysunek 6 - Krzywa uczenia modelu SVC



Rysunek 7 - Krzywa uczenia modelu MLP

8. Bibliografia

1. <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>
2. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
3. https://scikit-learn.org/stable/modules/neural_networks_supervised.html#multi-layer-perceptron
4. https://en.wikipedia.org/wiki/Limited-memory_BFGS
5. https://en.wikipedia.org/wiki/Stochastic_gradient_descent
6. [https://pl.wikipedia.org/wiki/K_najbliższych_sasiadów](https://pl.wikipedia.org/wiki/K_najbli%C5%9Bszych_sasiad%C3%B3w)
7. <https://scikit-learn.org/stable/modules/neighbors.html#nearest-centroid-classifier>
8. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html>
9. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html#sklearn.neighbors.BallTree>
10. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html#sklearn.neighbors.KDTree>
11. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
12. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
13. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
14. https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#sphx-glr-auto-examples-model-selection-plot-precision-recall-py
15. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
16. https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html#sphx-glr-auto-examples-model-selection-plot-learning-curve-py
- 17.