WSI

Zadanie 4

Autor:
Michał Paradowski

19.04.2024 (V1.0)

Contents

1 Zadanie

Celem zadania jest wykorzystanie modelu SVM oraz Drzew decyzyjnych w klasyfikacji danych. Dataset zawiera informacje na temat win i ich przynależności do konkretnej klasy. W trakcie implementacji algorytmów polecam wykorzystanie biblioteki scikit-learn. W celu wykonania zadania, muszą Państwo:

• pobrać, odpowiednio wczytać dane.
• wytrenować modele korzystając z walidacji krzyżowej (należy podzielić zestaw danych na 4 części. Każda część w kolejnej iteracji uczenia będzie pełnić rolę zbioru testowego)
• Dokonać oceny skuteczności wytrenowanego modelu korzystając z następujących metryk: Accuracy, Precision, Recall, F1 Score

Implementując algorytm SVM oraz drzewa decyzyjne możemy zmieniać wiele parametrów. Dla każdego z nich należy przetestować ich wpływ na wyniki:

SVM:
• Funkcja jądra (kernel)
• Siła regularyzacji
• Liczba iteracji (przy analizie wpływu tego parametru trzeba znacząco zmniejszyć parametr tol, którego domyślna wartość to 1e-3. Parametr tol służy jako kryterium zatrzymania procesu uczenia algorytmu)

Drzewa decyzyjne:
• Maksymalna głębokość drzewa
• Kryterium używane do oceny jakości
• Strategia podziału węzła

Implementacja tych algorytmów w bibliotece scikit-learn zawiera również parametr randomState odpowiadający za kontrolowanie losowości.Wszystkie pomiary należy powtórzyć 3 krotnie z różnym parametrem randomState aby mieć 3 oszacowania skuteczności każdego modelu.

Sprawozdanie
W sprawozdaniu należy umieścić opis działania algorytmów i zestawienie wyników skuteczności algorytmów. Proszę również o dołączenie wykresów przedstawiających jak zmienia się skuteczność modelu w zależności od zmiany wybranego parametru liczbowego takiego jak np. liczba iteracji w przypadku SVM.

2 Algorytmy

2.1 Support Vector Machine (SVM)

Support Vector Machine (SVM) jest popularnym modelem uczenia maszynowego, który jest wykorzystywany zarówno do klasyfikacji, jak i regresji (w tym zadaniu wykorzystamy go do klasyfikacji). Celem jest znalezienie hiperpłaszczyzny optymalnie dzielącej przestrzeń cech, aby maksymalizować margines między klasami.

2.1.1 Opis algorytmu

Niech $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)$ będą danymi treningowymi, gdzie $\mathbf{x}_i$ to wektor cech, a $y_i \in \{-1, 1\}$ to etykieta klasy. SVM szuka hiperpłaszczyzny opisanej przez $\mathbf{w} \cdot \mathbf{x} + b = 0$, gdzie $\mathbf{w}$ to wektor wag, b to przesunięcie (bias).
Optymalizujemy problem:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{n} \xi_i$$

przy ograniczeniach:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

gdzie ξ_i to marginesy odstępstwa, a C to parametr regularyzacji. Pierwsza część formuły ($\frac{1}{2}||w||^2$) reprezentuje 'złożoność' rozwiązania. Chcemy by długość wektora w była możliwie najmniejsza by uniknąć niepotrzebnej złożoności rozwiązania i możliwego overfittingu. $C \sum_{i=1}^{n} \xi_i$ odpowiada zaś karom za odstępstwa od podziału i jest regulowana przez parametr C.
Wartości $\mathbf{w}$ i b obliczane są przy użyciu metody optymalizacji, takiej jak metoda gradientu prostego lub metoda SMO (Sequential Minimal Optimization), minimalizując powyższe równanie.

W przypadku danych analizowanych w tym zadaniu musimy rozdzielić wina na 3 różne klasy. W takim przypadku tworzymy 3 różne instancje modelu SVM i w każdej z nich stosujemy strategię One-vs-Rest, (OvR). Polega ona na wybraniu jednej z trzech klas a następnie potraktowaniu dwóch pozostałych jako jednej. Tak więc każda instancja dzieli nam dane na dwie części. Tak naprawdę wystarczyły by dwie instancje algorytmu. Wtedy na podstawie wyników dwóch sprawdzeń moglibyśmy jednoznacznie określić do której spośród trzech możliwych klas należy wektor wejściowy.

2.1.2 Parametry

Algorytm SVM posiada kilka istotnych parametrów:

- C: Parametr regularyzacji kontrolujący wagę marginesów i błędów klasyfikacji. Duże wartości C prowadzą do mniejszego marginesu, co może prowadzić do overfittingu, zaś małe, do niedokładnego dopasowania modelu.
- Kernel: Określa funkcję jądra używaną do przekształcenia danych wejściowych do przestrzeni cech o wyższej wymiarowości. Popularne funkcje jądra to np. liniowe, wielomianowe, RBF (Radial Basis Function).
- Liczba iteracji: Określa maksymalną liczbę iteracji, po której algorytm zakończy działanie, jeśli nie zostanie osiągnięta zbieżność.
- Tolerancja: Określa warunek zbieżności, czyli maksymalną tolerowaną różnicę między wartościami funkcji kosztu w kolejnych iteracjach.

2.2 Drzewa decyzyjne

Drzewa decyzyjne są popularnym modelem uczenia maszynowego, który także może być wykorzystywany zarówno do klasyfikacji, jak i regresji. Algorytm budowy drzewa decyzyjnego polega na rekurencyjnym podziale przestrzeni cech na podprzestrzenie, aby ostatecznie uzyskać zbiory, które są jednorodne pod względem etykiet klas lub wartości docelowych.

2.2.1 Opis algorytmu

Niech $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)$ będą danymi treningowymi, gdzie $\mathbf{x}_i$ to wektor cech, a y_i to etykieta klasy lub wartość docelowa.
Algorytm budowy drzewa decyzyjnego składa się z następujących kroków:

1. Podział węzła: Algorytm wybiera podział przestrzeni cech w węźle drzewa na podstawie wybranego kryterium oceny jakości, takiego jak współczynnik Giniego lub entropia. Podział ten ma na celu maksymalizację jednorodności podprzestrzeni wynikowych.

2. Rekurencyjna budowa drzewa: Proces podziału jest powtarzany rekurencyjnie dla każdego z nowo powstałych węzłów, aż do spełnienia pewnych warunków stopu, takich jak maksymalna głębokość drzewa, minimalna liczba próbek w liściu lub brak poprawy jakości podziału.

2.2.2 Parametry

Algorytm drzewa decyzyjnego ma kilka istotnych parametrów:

- Maksymalna głębokość drzewa: Określa maksymalną liczbę poziomów w drzewie, co zapobiega nadmiernemu dopasowaniu (overfittingowi) do danych treningowych.
- Kryterium używane do oceny jakości: Określa metrykę używaną do oceny jakości podziału węzła, na przykład współczynnik Giniego lub entropię.
- Strategia podziału węzła: Określa, jakie kryterium jest używane do wyboru najlepszego podziału węzła, na przykład "najlepszy" (best) lub "losowy" (random).

Wykonam sumarycznie 6 eksperymentów w których zmierzę wpływ każdego parametru na wyniki obu algorytmów. W danych z zadania znajdują się informacje o winach podzielonych na 3 klasy na podstawie 13 różnych parametrów. To, co te parametry przestawiają nie ma żadnego znaczenia dla działania algorytmu więc nie będę się w to zagłębiał.

3 Max iterations

Parametr max iterations, jak sama nazwa wskazuje określa największą liczbę iteracji przez którą algorytm może działać. W celu przetestowania wpływu tego parametru zmniejszyłem parametr tolerancji aby wymusić na algorytmie wykorzystanie wszystkich iteracji. Poniżej zaprezentuję na wykresach wyniki eksperymentu z użyciem funkcji jądra odpowiednio rbf i wielomianowej. ponieważ na funkcję liniową liczba iteracji ma nieco mniejszy wpływ.

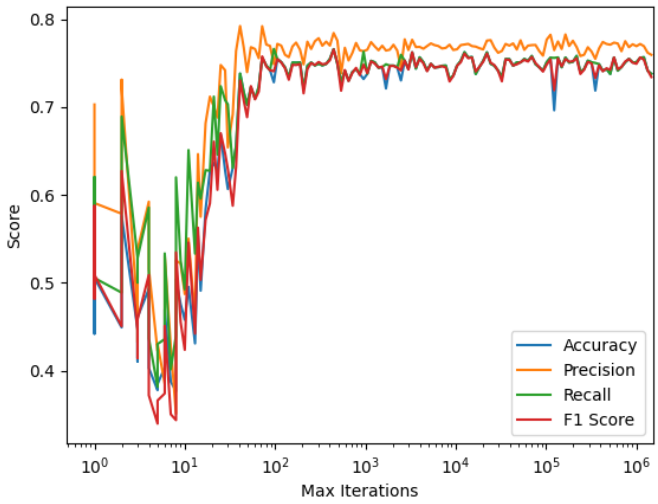


Figure 3.1: (algorytm z funkcją jądra rbf)

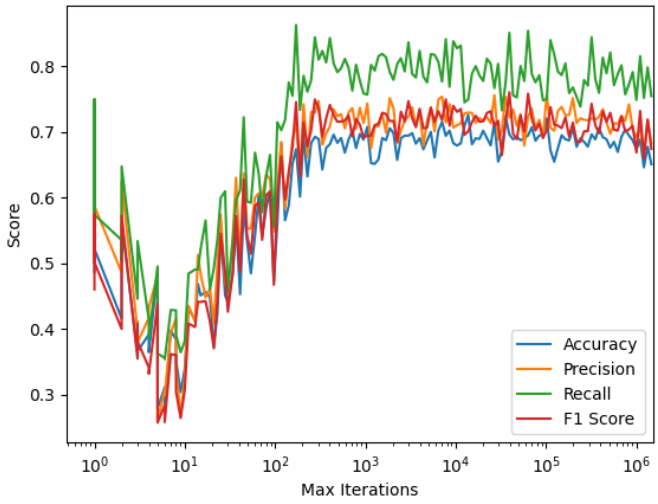


Figure 3.2: (algorytm z funkcją jądra wielomianową)

Widzimy że dla liczby iteracji bliskich zeru wyniki wszystkich wycen wskazują zgodnie z oczekiwaniami na rozkład losowy. następnie wyceny maleją co mówi nam że na samym początki algorytm przewiduje całkowicie błędnie ze względu na uwzględnianie niepoprawnych danych. Następnie wyniki poprawiają się i stabilizują się przy około 100 iteracjach dla obu funkcji jądra. Baza danych jest mała więc niewielka ilość iteracji jest spodziewana. Gdyby dane były bardziej skomplikowane, przy większej liczbie iteracji moglibyśmy spodziewać się zjawiska oferfittingu. Tak czy inaczej stosowanie więcej niż 200 iteracji nie jest tutaj wskazane ze względu na szybkość działania algorytmu.

4 Parametr C

Parametr C określa to, jak dokładnie model musi odzwierciedlać dopasowywać dane treningowe by otrzymać wysoką wycenę. Zbyt niska jego wartość prowadzić może do niedostatecznej dokładności algorytmu, zaś zbyt wysoka do bardziej skomplikowanej funkcji oddzielającej i do oferfittingu do danych. Jak zauważyliśmy we wcześniejszych eksperymentach, overfitting w przypadku tych danych nie wydaje się być problemem, więc spodziewamy się zobaczyć wyniki które go nie pokażą. Oto wyniki eksperymentu przeprowadzonego by to sprawdzić:

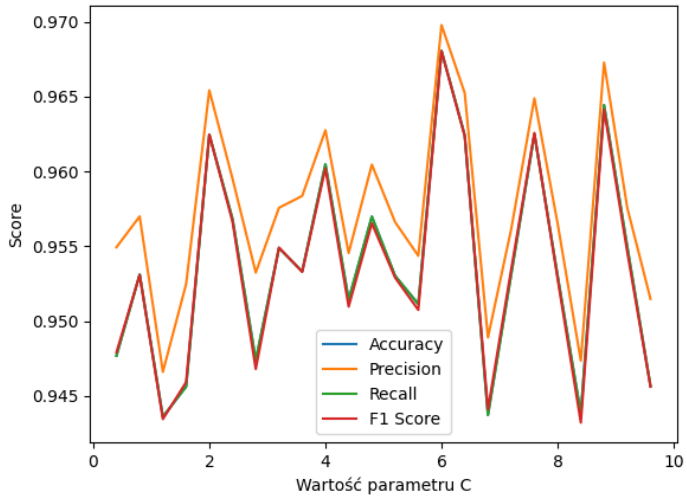


Figure 4.1: (algorytm z funkcją jądra wielomianową)

Mimo wielokrotnych prób z różnymi funkcjami jądra wpływ parametru C w przypadku tych konkretnych danych okazał się nikły. Może być to spowodowane wieloma różnymi czynnikami. Ten najbardziej prawdopodobny to dobrze separowalne dane. Jeśli dane są dobrze separowalne to kara za złą separację jest niewielka niezależnie od wartości C.

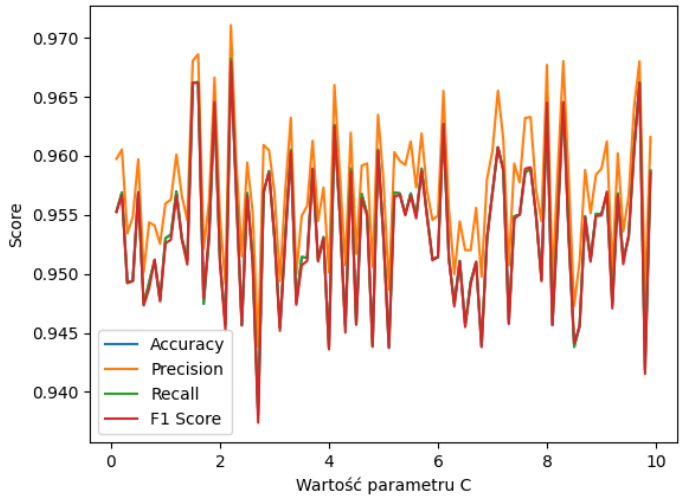


Figure 4.2: (algorytm z funkcją jądra wielomianową)

5 Funkcja jądra

Biblioteka scikit-learn udostępnia 3 podstawowe funkcje jądra:

- linear - funkcja liniowa
- poly - funkcja wielomianowa
- rbf - radial basis function
- sigmoid

Parametr ten decyduje o rodzaju funkcji która wyznacza granice między klasami. funkcja może być liniowa, wielomianowa lub rbf. Poniżej wyniki eksperymentów z parametrem C = 1, max iter = -1 i kolejno kernell='linear', 'poly', 'rbf', 'sigmoid'.

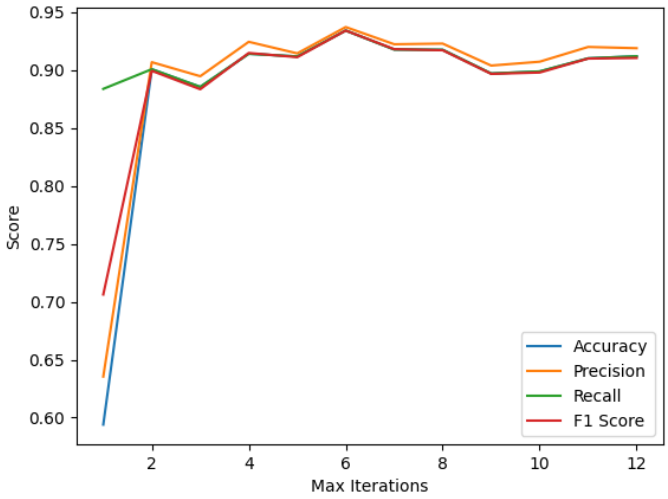
Kernell:	linear	poly	rbf	sigmoid
accuracy score:	0.9458	0.6686	0.6722	0.1983
precision score:	0.9536	0.6486	0.6935	0.21395
recall score:	0.9458	0.7210	0.7476	0.3658
f1 score:	0.9463	0.6686	0.7035	0.2478

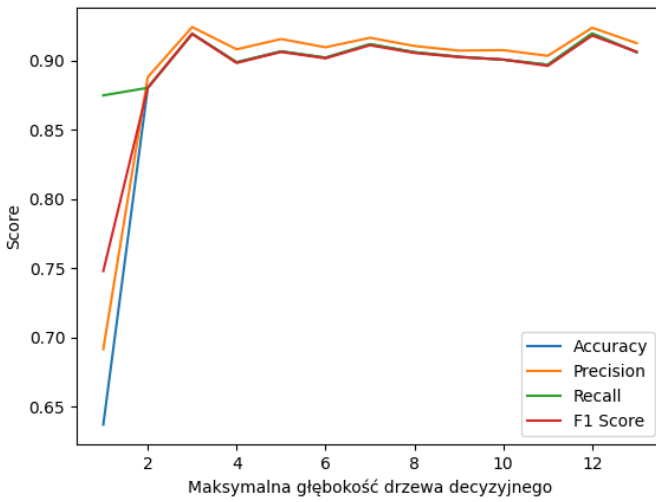
Table 5.1: Wyniki eksperymentu

Jak widać najlepszym wyborem okazała się funkcja liniowa. Jest to zgodne z oczekiwaniami jako że doszliśmy do wniosku że parametry dość zgodnie dzielą wina na klasy. Funkcje wielomianowa i Rbf były gorsze, ale nadal sprawdziły się dobrze, zaś funkcja sigmoid okazała się w przypadku tych danych nieskuteczna. Wyniki te pokazują jak ważne jest empiryczne sprawdzenie i dobranie odpowiedniej funkcji jądra tym bardziej jeśli nie znamy rozkładu danych.

6 Głębokość Drzewa i kryterium wyboru

Głębokość drzewa decyduje o liczbie węzłów decyzyjnych. Gdy jest zbyt mała, może prowadzić do niedostatecznego dopasowania modelu do danych, gdy zaś jest zbyt duża, może prowadzić do overfittingu. Każde wino w bazie danych z zadania ma 13 parametrów. Wynika z tego że maksymalna sensowna liczba węzłów to 13, gdyż każdy węzeł obsługuje jeden parametr. Poniższe wykresy przedstawiają wyniki wszystkich wycen w zależności od głębokości drzewa odpowiednio dla kryterium 'best' i 'random'. (Kryterium 'best' polega na wyborze parametru i podziału którego wpływ na zmianę entropii jest największy, zaś random, na wyborze losowego parametru i podziału)





Jak widać, dla kryterium 'random' wszystkie 4 wyceny osiągnęły swoje maksima przy maksymalnej głębokości drzewa równej 6. Później prawdopodobnie zaczął się uwydatniać overfitting modelu. Ciekawą obserwacją może być fakt, że już głębokość 2 przyniosła tam dość precyzyjne rezultaty. Wynikać to może z faktu że w danych występuje mało chaosu i większość parametrów skutkuje podobnym podziałem na klasy. Tę tezę popierają wyniki dla kryterium 'best', gdzie już jedynie głębokość 3 skutkuje najlepszymi wynikami. Widać więc że wystarczy rozpatrzeć jedynie 3 najbardziej znaczące parametry by uzyskać bardzo dobry wynik.

7 Parametr criterion

Parametr criterion odpowiada za sposób oceny podziału węzła. To właśnie funkcja tutaj wybrana mówi nam który spośród uzyskanych możliwych podziałów jest najkorzystniejszy. Biblioteka scikit-learn udostępnia nam do wyboru 3 takie funkcje:

1. gini - $G(t) = 1 - \sum_{i=1}^{J} p(i|t)^2$, gdzie J - liczba klas, $p(i|t)$ - prawdopodobieństwo na wystąpienie klasy i w węźle t.
   Funkcja ta mierzy więc 'czystość' podziały. Im mniejsza liczba klas w węźle tym lepiej.

2. entropy - $H(t) = -\sum_{i=1}^{J} p(i|t) \log_2(p(i|t))$.
   Ta funkcja mierzy entropię podziału. Im mniejsza entropia (niepewność danych), tym lepiej.

3. log-loss - $\log\_loss = -\frac{1}{N} \sum_{i=1}^{N} (y_i \log(p_i) + (1 - y_i) \log(1 - p_i))$, gdzie N to liczba próbek, $y_i$to prawdziwa etykieta klasy, a $p_i$to przewidywane prawdopodobieństwo przynależności do klasy 1.

Oto wyniki eksperymentu z parametrami splitter = 'best' i maksymalna głębokość = 6.

Criterion:	gini	entropy	$\log_l oss$
accuracy score:	0.9102	0.9116	0.8969
precision score:	0.9174	0.9062	0.9007
recall score:	0.9102	0.9062	0.8969
f1 score:	0.9086	0.9062	0.8966

Table 7.1: Wyniki eksperymentu

Jak widać najlepiej wypadło kryterium 'gini', niewiele gorzej 'entropy' a 'log-loss' już zauważalnie gorzej lecz nadal dobrze. Podobieństwo w wynikach 'gini' i 'entropy' Wynika prawdopodobnie z podobieńswa ich działania. Obie względniają prawdopobieństwa na zakłucenia w danych, jednak z trochę innymi wagami. Najlepiej w tym przypadku skorzystać więc z funkcji 'gini'

List of Figures