WSI - ćwiczenie 7

Modele bayesowskie

Levchenko Sofiia, nr. indeksu 308996

10.06.2002

Wstęp

Celem tego ćwiczenia było zaimplementowanie naiwnego klasyfikatora Bayesa, a następnie wykorzystać tworzony algorytm do stworzenia klasyfikatora dla zbioru danych iris. Do zbadania należało wykorzystać algorytm n-krotnej walidacji krzyżowej i następnie porównać wynik z otrzymanym na wydzielonym wcześniej zbiorze testowym

Struktura projektu

Ze względu na specyfikę wykonywanego zadania, zdecydowałyśmy się, że struktura projektu będzie wyglądała w następujący sposób:

- *bayes_classifier.py* – implementacja klasy, reprezentującej naiwny klasyfikator bayesowski, dana klasa zawiera funkcje pomocnicze do obliczania prawdopodobieństwa wystąpienia klasy pod warunkiem posiadania określonych atrybutów (na podstawie rozkładu normalnego) i następnie obliczania wartości predykcji zarówno jak dla k-krotnej walidacji krzyżowej, tak i podziału zbioru na walidacyjny i testowy w sposób zwykły
- *experiments.py* - funkcje z definicjami eksperymentów przeprowadzonych w trakcie zadania
- *main.py* - zawiera wywołania eksperymentów przeprowadzonych w trakcie realizacji zadania
- *helpers.py* – zawiera funkcje pomocnicze do sporządzenia wykresów (wizualizacji uzyskanych wyników), obliczania dokładności wyniku uzyskanego przez klasyfikator

W ramach projektu były wykorzystywane różne biblioteki zewnętrzne, ułatwiające wykonanie zadania:

- *numpy* - do działań na różnych strukturach danych oraz przeprowadzenia na nich obliczeń/działań matematycznych.
- *pandas* - do przetwarzania danych w celu wizualizacji wyników
- *matplotlib* - biblioteka ułatwiająca tworzenie wykresów
- *seaborn* – biblioteka ułatwiająca wizualizację uzyskanych wyników
- *sklearn* - biblioteka ułatwiająca wizualizację uzyskanych wyników (matryca błędów)

Decyzje projektowe

W przypadku danego zadania, model naiwnego klasyfikatora bayesowskiego został zaprezentowany w postaci klasy BayesClassifier.

Głównymi metodami tej klasy, umożliwiającymi poprawne działania modelu są:

- *get_statistics* – metoda ta pozwala na obliczanie wartości średniej oraz wariancji wartości liczbowej określającej atrybut (kolumna badanego zbioru danych)
- *get_prior* oraz *get_posterior* – do określenia prawdopodobieństwa wystąpienia pewnej klasy oraz prawdopodobieństwa warunkowego posiadania pewnej wartości atrybutu pod warunkiem tego, że element posiada określoną klasę
- *density* – do określenia prawdopodobieństwa wystąpienia pewnej klasy, korzystając z rozkładu normalnego

Inne metody, wykorzystywane w danej klasie przeznaczone są do predykcji klasy na podstawie obliczonych prawdopodobieństw oraz podziału zbioru na walidacyjny i testowy w przypadku k-krotnej walidacji krzyżowej.

Opis wykonanych eksperymentów

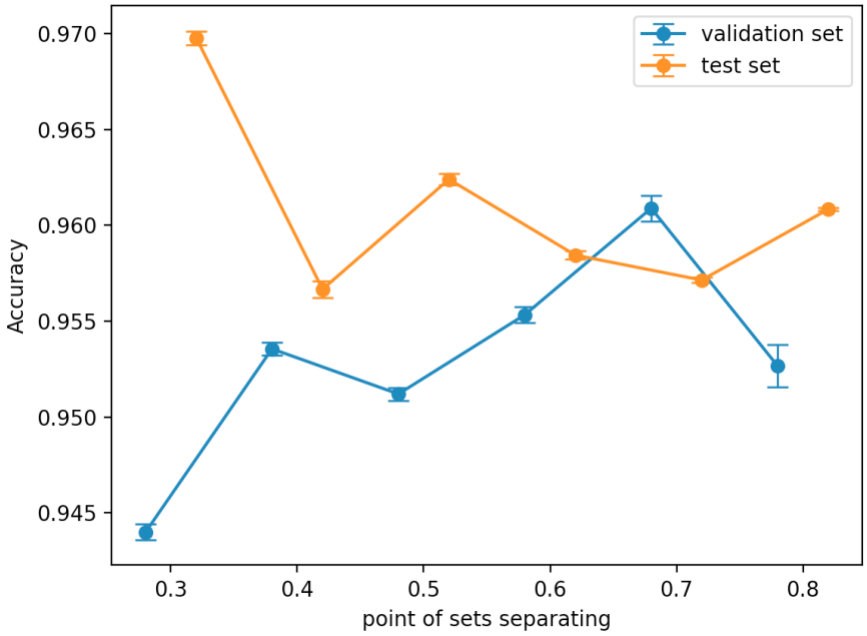
Na sam początek, został zbadany wpływ punktu podziału zbioru danych na zbiór walidacyjny i trenujący, aby następnie porównać otrzymane wyniki z uzyskanymi podczas k-krotnej walidacji krzyżowej. W tym celu została obliczona dokładność uzyskiwanych wyników dla 6 punktów podziału. Dla każdego punktu podziału zostało model był uczony 50-krotnie, gdzie za każdym razem zbiór danych był shuffle'owany, aby uzyskać bardziej wiarygodny wynik.

Następnie na podobnej zasadzie został zbadany wpływ liczby k przy k-krotnej walidacji krzyżowej oraz uzyskany wynik został porównany z uzyskanym wcześniej.

Dla najlepszego uzyskanego podczas badań punktu podziału (w przypadku podziału zbioru na walidacyjny i trenujący) została zwizualizowana matryca błędów dla obu zbiorów oraz ten wynik też został zwizualizowany w postaci wykresu słupkowego.

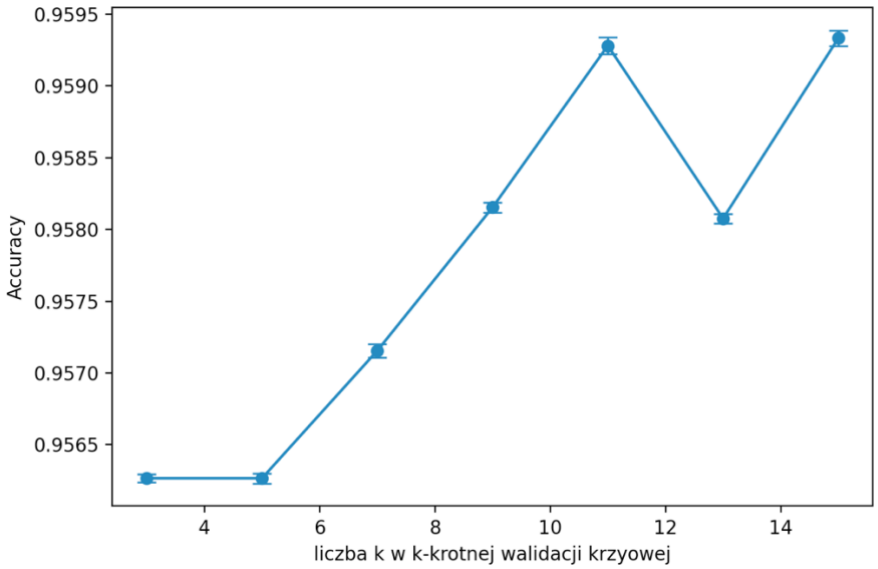
Wyniki

Wynik, uzyskany po podziale zbioru danych na walidacyjny i trenujący ma dość typowy wygląd – dla małego lub zbyt dużego zbioru trenującego prawdopodobieństwo uzyskania dobrego wyniku dla zbioru walidacyjnego nie jest najwyższe (chociaż i tak a bardzo dobrą jakość – prawidłowość uzyskanych wyników jest powyżej 0.9), przez zbyt duże dostosowywanie modelu do danych trenujących.

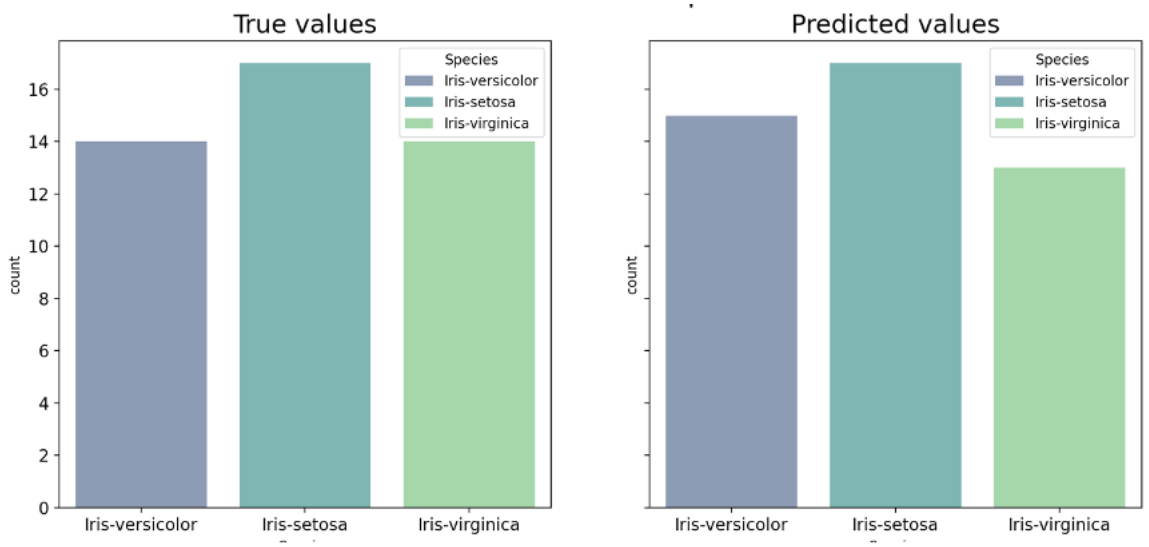


Jak widać na powyższym wykresie – najlepsze wyniki jednak uzyskujemy dla punktu podziału 0.7, wtedy jeszcze nie ma nadmiernego dopasowania, jednak kompendium wiedzy modelu jest wystarczający, aby wyniki dla zbioru walidacyjnego były lepsze niż dla zbioru trenującego.

Po przeprowadzeniu k-krotnej walidacji krzyżowej, średnio jakość uzyskiwanych wyników była na poziomie 0.95-0.96, co jest nieco mniejszą wartością, niż uzyskaną dla poprzedniego eksperymentu (o około 0.05-0.1), co świadczy o tym, że ogólnie model jest zaprojektowany dobrze.

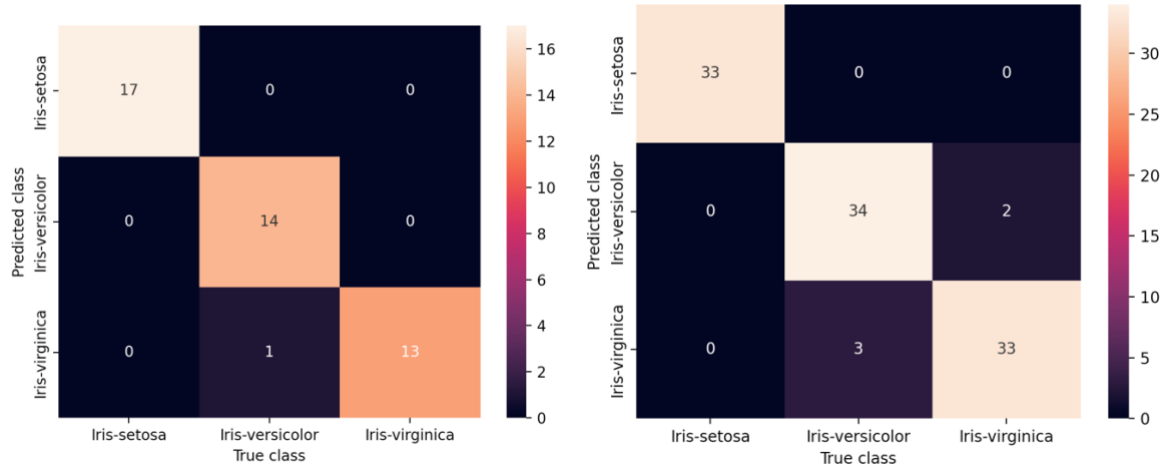


Jeżeli chodzi o wpływ wartości k na otrzymywany wynik, to możemy zauważyć, że wartość, określająca jakość klasyfikatora nie zmienia się jakoś mocno (w zakresie +-0.003), jednak dla wystarczająco dużych wartości k (jednak nie za bardzo) uzyskiwany wynik jest lepszy.



Jeżeli chodzi o rozpoznawanie klas przez zaimplementowany klasyfikator, to na wyżej umieszczonym rysunku możemy zauważyć, że klasyfikator bardzo dobrze sobie radzi z rozpoznawaniem klasy „iris-setosa”, jednak czasami potrafi mylić „iris-virginica” i „iris-versicolor”.

Tego samego wniosku możemy dojść, patrząc na matrycę błędów, umieszczoną poniżej (po prawej mamy wynik dla zbioru trenującego, a po prawej dla zbioru walidacyjnego).



Wnioski

Możemy zauważyć, że dla porównywalnie małej złożoności dokonywanych obliczeń, taki klasyfikator potrafi bardzo dobrze rozpoznawać klasy (dokładność około 0.96). Ten klasyfikator czasami potrafi mylić niektóre klasy ze sobą, jednak błędy takiego typu dokonywane przez model bardzo rzadko).

Nie zwracając uwagi na prostotę danej metody (przynajmniej jeżeli chodzi o proces implementacyjny), klasyfikator ten potrafi bardzo dobrze radzić sobie z różnego rodzaju problemami klasyfikacji.