Programowanie w językach skryptowych Python i linux Bash

Badanie działania i sprawności algorytmu KNN w projekcie

Filip Pawelec, Sławomir Potoczek

14 grudnia 2020

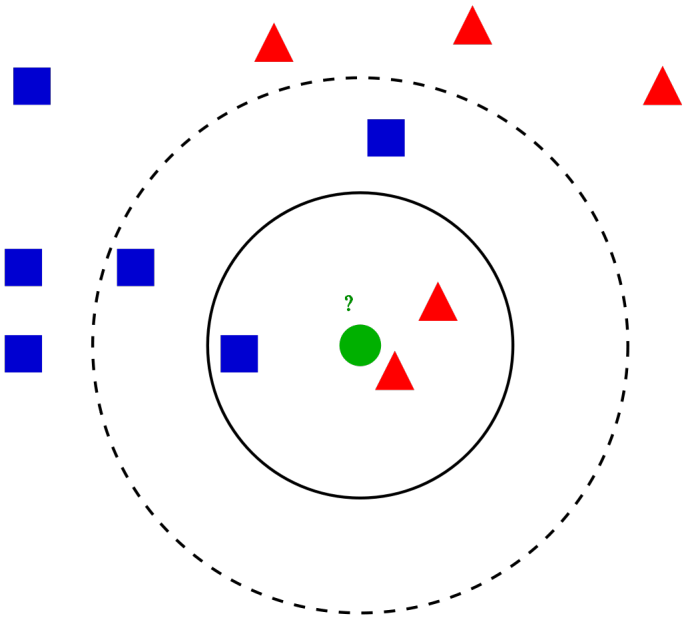
1 Krótkie omówienie algorytmu KNN

KNN (K Nearest Neighbours) jest jednym z podstawowych i prostszych algorytmów wykorzystywanych w *machine learningu*. Algorytm ten jest sztandarowym przykładem klasy algorytmów nazywanej *lazy learners*. Zaletą tego typu algorytmów jest nauka na bieżąco, to znaczy, że jeżeli zestaw danych treningowych zmienia się w sposób ciągły, zmiany te są w odpowiedni sposób brane pod uwagę przy każdym zapytaniu do systemu. W przypadku drugiej kategorii algorytmów, *eager learners*, aby nowe dane treningowe zostały poprawnie wzięte pod uwagę, musi nastąpić "rekompilacja" wyników treningu, co dla dużych objętości danych może powodować problemy z mocą obliczeniową.

Wadą algorytmu KNN, przy dużej objętości danych treningowych, jest obliczanie odpowiedzi na pytanie do systemu (wymaga większej mocy obliczeniowej), dlatego często dla dużych baz danych odpowiedzi na najczęściej zadawane pytania oblicza się z wyprzedzeniem, w godzinach kiedy serwery są najmniej obciążone.

2 Wpływ wartości parametru k na działanie algorytmu oraz jego *accuracy*, *f1 score* i *confusion matrix*

k jest parametrem mówiącym o ilości najbliższych punktów/zdarzeń/wyników, do których będzemy porównywać nowy punkt/zdarzenie/wyniki. k musi być precyzyjnie określone, gdyż jeżeli będzie zbyt małe lub zbyt duże spowoduje to błędnie interpretowane wyniki - zmniejszenie *accuracy* i *f1 score*.



Obrazek 1: dla $k = 3$ (okrąg ciągły) zielona kropka zostanie zaklasyfikowana do czerwonych trójątów, natomiast dla $k = 5$ (okrąg przerywany) będą to już niebieskie kwadraty

Najbardziej optymalnym równaniem na otrzymanie k jest $k = \sqrt{n}$, gdzie n jest całkowitą ilością danych. W przypadku gdy k jest wartością parzystą, należy dodać lub odjąć 1 aby uniknąć sytuacji, w której liczba sąsiadów z różną klasyfikacją jest taka sama.

Przy zastosowaniu tego równania ($k = \sqrt{n}$) wartość k dla testowego datasetu wynosi 11. Tak prezentują się najważniejsze parametry algorytmu:

```
Confusion matrix:
[[94 13]
 [15 32]]
Accuracy:  0.8181818181818182
F1 score: 0.6956521739130436
```

Obrazek 2: Parametry dla $k = 11$

Dla porównania oraz potwierdzenia odzwierciedlenia opisanej wyżej teorii wyniki:

- dla $k = 9$:

```
Confusion matrix:
[[92 15]
 [16 31]]
Accuracy:  0.7987012987012987
F1 score: 0.6666666666666667
```

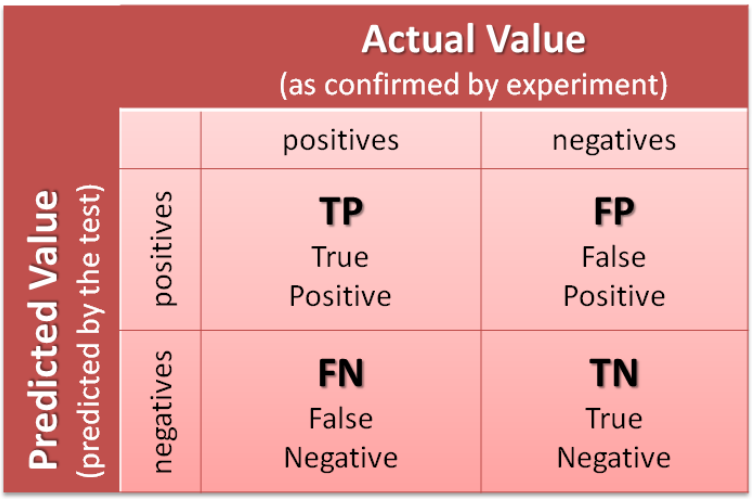
Obrazek 3: Parametry dla $k = 9$

- dla $k = 13$:

```
Confusion matrix:
[[95 12]
 [16 31]]
Accuracy:  0.8181818181818182
F1 score: 0.6888888888888888
```

Obrazek 4: Parametry dla $k = 13$

Jak widzimy na powyższych obrazkach, najwyższe wartości *accuracy* i *f1 score* przypadają dla $k = \sqrt{n}$. Warto spojrzeć także na *confusion matrix*, która mówi nam ile dokładnie przwidywań było *True* (prawdziwych), a ile *False* (nieprawdziwych), rozdzielając je na *Positive* (stwierdzono cukrzycę) oraz *Negative* (nie stwierdzono cukrzycy). Dla ułatwienia odczytania wartości *confusion matrix* przedstawionych powyżej, warto zaznajomić się z poniższym obrazkiem:



Obrazek 5: Przewodnik po terminologii *confusion matrix*

Najważniejszą wartością dla algortymów stosowanych w medycynie jest lewa dolna wartość *confusion matrix*, która przedstawia ilość przypadków *False Negative*, czyli niestwierdzonych chorych - tych, których za wszelką cenę stara się unikać. Dlatego pracując nad algorytmem w pierwszej kolejności dąży się do tego, aby wartość ta była równa 0. Patrząc na obrazki z parametrami algorytmu dla różnych k widzimy, że także w tym przypadku dla stwierdzonego wcześniej optymalnego k wartość FN (*False Negative*) *confusion matrix* jest najmniejsza.

3 *Feature scaling*

Jednym z zagadnień które trzeba rozważyć pracując nad algorytmem KNN jest branie pod uwagę jakiego rzędu wielkości są parametry wejściowe i w jaki sposób wpływają na wyniki klasyfikacji. Bardzo dobrą praktyką, stosowaną nawet pomimo niewielkiego rozrzutu rzędu wielkości danych, jest *feature scaling*. Polega to na przeskalowaniu wielkości na odpowiednio mniejszy zakres. Standardowo skaluje się parametr wejściowy licząc jego wartość średnią oraz odchylenie standardowe, a następnie przeskalowaną wartość oblicza się następująco:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

gdzie x to nieprzeskalowana wartość, μ to średnia, a σ to odchylenie standardowe. Dzięki temu zabiegowi kiedy algorytm będzie liczył kartezjańską odległość, nie będzie ona podyktowana przez dane wejściowe o największych rzędach wielkości.

Wpływ *feature scaling* na poprawność działania algorytmu zostały pokazane na poniższym obrazku, na którym znajdują się jego parametry bez zastosowania *feature scaling*:

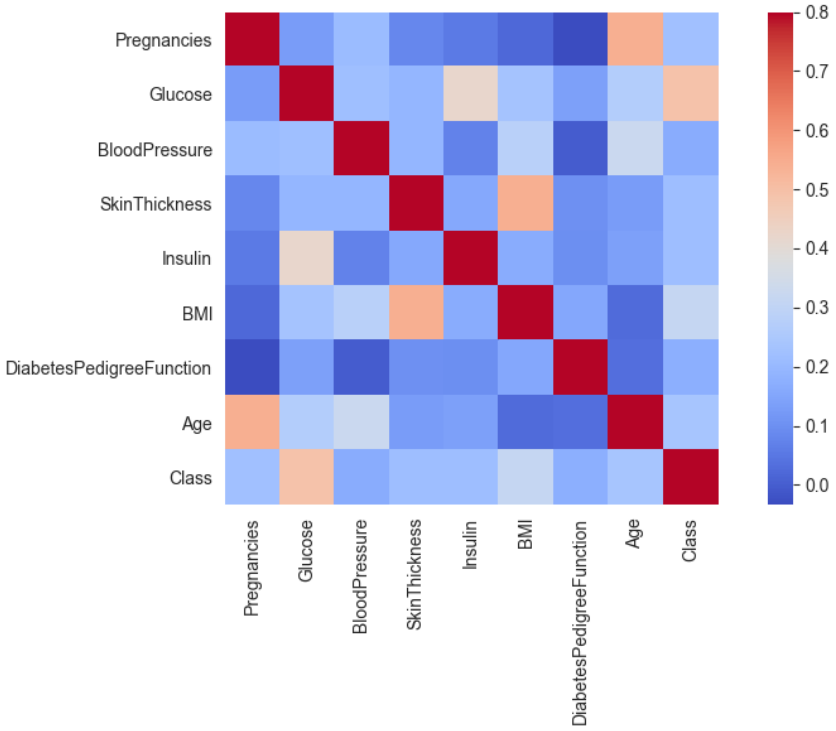
```
Confusion matrix:
[[86 21]
 [16 31]]
Accuracy:  0.7597402597402597
F1 score: 0.6262626262626262
```

Obrazek 6: Parametry bez *feature scaling*

Porównując powyższy obrazek do obrazka nr 2, na którym widać parametry algorytmu po zastosowaniu *feature scaling*, widać znaczną różnicę. Zarówno wartości *accuracy* i *f1 score* są zauważalnie wyższe na obrazku 2, niż obrazku powyższym. Także *confusion matrix* ulega poprawie, szczególnie jej lewy górny róg - wartość TP (*True Positive*).

4 *Korelacje*

Jednym z zagadnień które trzeba rozważyć pracując nad algorytmem KNN jest branie pod uwagę w jakim stopniu dany parametr wejściowy wpływa na wynik klasyfikacji. Żeby ustalić taką zależność - stworzyć funkcję wagową - liczone są współczynniki korelacji pomiędzy parametrami (każdy z każdym). W badanym przez nas przypadku wyniki tych obliczeń można obserwować na poniższym obrazku:



Obrazek 7: Współczynniki korelacji dla parametów wejściowych przedstawione na heatmapie

Współczynniki te są symetryczne względem diagonali, ponieważ dane wejściowe nie są zespolone (wówczas byłyby sprzężone). Sama diagonala to jest współczynnik korelacji danego parametru z samym sobą, dlatego wynosi on najwięcej.