Podobnie jak przy survivalu gównoburzy z baz, tak i tutaj nie gwarantuję poprawności podanej treści, nawet nie wiem, czy ruszy, bo nie napisałem tego jeszcze, ale ze strzępków wiedzy i męskiej intuicji wydaje mi się, że powinno zadziałać

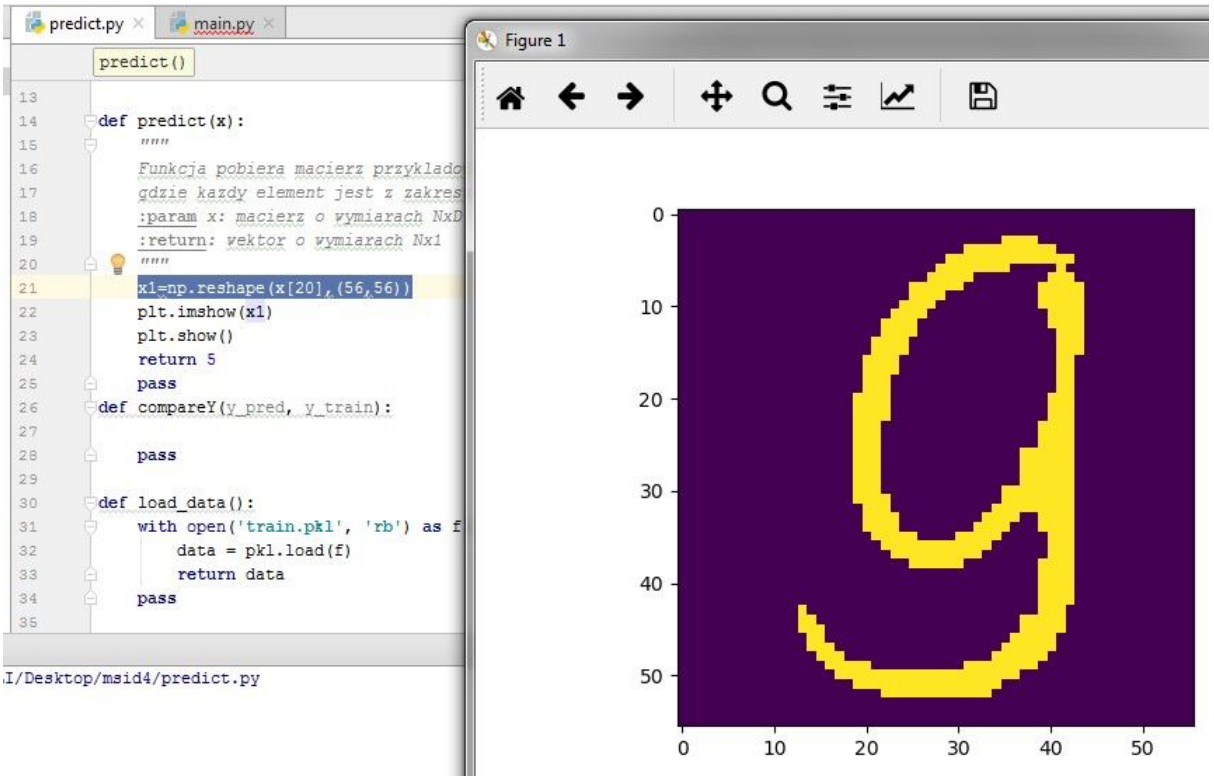
No więc chcemy tym KNN-em zrobić nasz algorytm, tylko nie wiemy, co jest czym, ani jak to ogarnąć? Myślę, że mniej więcej pojąłem ideę oraz co jest czym w zadaniu i postaram się to wyłożyć w miarę zrozumiale. Jak ktoś zauważy błędy/stwierdzi, że to wszystko brednie, niech niezwłocznie do mnie napisze, żebym to poprawił/zdjął i nie robił sobie więcej wstydu/krzywdził innych fałszywymi informacjami.

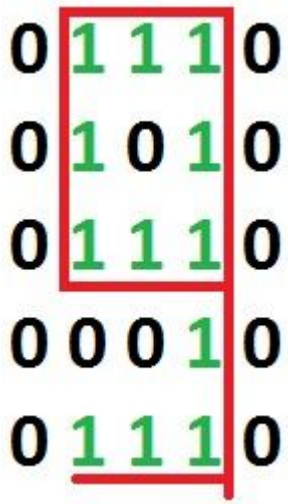
Zacznijmy od nazewnictwa:

Ypred - predykcja nowej klasy, inaczej jaki to najprawdopodobniej będzie znaczek - to sami musimy wyprodukować na podstawie tego co otrzymujemy na starcie

Xnew - nowe dane, dla których musimy klasę predykować (**Ypred**) , inaczej nowe obrazki ze znaczkami zapisane w macierzy 56x56 za pomocą 0 i 1. Macierz 56x56 w tych danych reprezentowana jest jako długi wektor 1x3196 czy coś koło tego (na koniec pierwszego doklejamy drugi wiersz, potem trzeci do całości itd. taka binarna stonoga) - **Xnew** na starcie jest podane

Ogólnie w programiku naszym w predict(**x**) <- ten parametr **x** to jest właśnie to **Xnew**, dla którego musimy odgadnąć y czyli klasę, czyli jaki to znaczek :D





^ Tak to w uproszczeniu wygląda jakbyście sobie wyświetlili zawartość macierzy jako tekst.

0 1 1 1 0 | 0 1 0 1 0 | 0 1 1 1 0 | 0 0 0 1 0 | 0 1 1 1 0

Yreal – faktyczne klasy naszych wierszy z **Xnew** (Nie wiemy, czy to 1, czy też 7, a ta klasa jednoznacznie podaje, że to 7) - to mamy podane na starcie

Ytrain - klasy dla **Xtrain** faktyczne - podane na starcie

Xtrain - dane, do których będziemy Xnew porównywać, aby określić jaki to znak(klasa) - to też na starcie jest podane.

Xtrain - **Ytrain**
Xnew - **Yreal** <> **Ypred**

Czyli spinając to w całość: Mamy **Xnew, Xtrain, Ytrain, Yreal** oraz **Ypred.**

Naszym zadaniem jest wyznaczyć **Ypred** i porównać z **Yreal**, aby ustalić jaki będzie błąd.

Jak zatem tego dokonać?

Jak większość zapewne słyszała, w KNN-ie wyznaczamy nową klasę na podstawie **k** najbliższych sąsiadów.

Mamy 3 klasy: kółko, krzyżyk, kwadrat.

Na takim wykresiku wpada nam nowy punkt (**Xnew**), dla którego mamy wyznaczyć klasę (**Ypred**).

Wówczas naszym zbiorem **Xtrain** będą **wszystkie punkty** na diagramie, **Ytrain** zaś określa, jakiego rodzaju jest dany punkt ze zbioru **Xtrain (**kółko, krzyżyk, kwadrat), inaczej jego klasę.

Aby dokonać predykcji - dla nowego punktu musimy wyznaczyć coś takiego jak **odległość**. Na takim diagramie intuicyjnie widać, że to kółko z nowym punktem w centrum, w którym mieści się **k** punktów.

Odległość - to jest kluczowy parametr, dzięki czemu możemy stwierdzić, co kryje się w naszym otoczeniu.

W drugim zadaniu wspomniane było o odległości hamminga, jako różnicy między bitami w pierwszym i drugim zbiorze (Stonodze binarnej). Odległość Hamminga to po prostu pewna miara. Moglibyśmy użyć innego sposobu obliczania odległości (układ współrzędny, odległość między punktami chociażby).

Kolejny przykład, tym razem w naszych klimatach.

Dostajemy macierz/obserwacje **Xnew** i chcemy predykować klasę dla pierwszego jej elementu/obserwacji (Xnew[0]).

Pierwszy wiersz może wyglądać jakoś tak:

0 1 1 0 1 0 1 (1x7)

W naszym **Xtrain** posiadamy kilka obserwacji/znaków do rozpoznania wraz z przypisanymi klasami **Ytrain**.

Obliczmy hamminga dla pierwszych trzech wierszów Xtrain:

^ Tutaj się wkradł błąd, miało być 4

Innymi słowy XOR'ujemy oba wiersze, to co zostaje nam, sumujemy.

W pierwszym wierszu nasz Xnew różni się na 7-miu pozycjach.
1+1+1+1+1+1+1=7

W drugim na 2-ch.

Itd.

Więc w sumie wychodzą nam takie odległości:

7 2 5 5 4 4

Klasy Ytrain odpowiadające wierszom porównywanym z Xtrain:

5 4 4 3 3 5

I teraz dochodzimy do momentu z sortowaniem, przy którym pewnie większość nie wiedziała o co chodzi (łącznie ze mną :D).

Zapiszmy sobie te dane w postaci krotki (**Odległość**, Klasa)

Mamy więc:

(**7**, 5), (**2**, 4), (**5**, 4), (**5**, 3), (**4**, 3), (**4**, 5)

Teraz sortujemy te krotki wg. odległości, rosnąco. Wychodzi nam:

(**2**, 4), (**4**, 3), (**4**, 5), (**5**, 3), (**5**, 4), (**7**, 5)

I teraz wchodzi do gry nasze **k.**

dla k=3 mamy następujące klasy sąsiadów:

4, 3, 5

dla k=5

4, 3, 5, 3, 4

No i wybór naszej klasy polega na wybraniu cyferki, która najczęściej występuje.

W przypadku k=3 mamy remis, więc bierzemy klasę, która jest najbliższym sąsiadem (4)

// Faktycznie mój błąd. Doczytałem jeszcze trochę, w przypadku jak występuje remis, to bierzcie klasę najbardziej z lewej/najbliższego sąsiada

// (W powyższym przypadku 4)

jednak dla k=5 widać, że klasa nr. 4 jest zwycięzcą.

// (Poprawione). Dla k=4 mielibyśmy 4,3,5,3 <- 2x3, zatem 3 byłoby naszą predykowaną wartością.

Inny przykład:

(**2**,3), (**3**,4), (**3**,3), (**5**,5), (**6**,1)

Pogrubione - posortowane odległości

Podkreślone - klasy powiązane z odległościami

Dla k=2 mielibyśmy klasy 3 i 4 czyli znowu randomowo (generalnie przyjmijcie zasadę, żeby przy remisie brać klasę najbliższą, w tym przypadku 3)

Dla k=4 - 3 4 3 5 -> wygrywa 3

Więc dla k=5 nasze **Ypred** będzie równe 4.

No i analogicznie postępujemy dla każdej kolejnej obserwacji/znaczku w **Xnew.**

Na końcu nasze **Yreal** wraca do gry. Dzięki niemu możemy wyznaczyć te błędy w predykcjach.

Załóżmy, że dla k=5

Ypred : 4 3 2 4 5 3

Ytrain : 4 2 2 3 5 3

Zatem przy 2 predykcjach się pomyliliśmy.

Możemy teraz spróbować zmienić wartość k na np. 2

Ypred: 4 3 1 3 2 1

Ytrain : 4 2 2 3 5 3

Tutaj różnica w 4 predykcjach, jeszcze większa lipa.

Ogółem sztuka sprowadza się do ogarnięcia k najlepszego. Właśnie w tym celu dokonuje się tej selekcji modelu, dla której nazwa "Selekcja hiperparametrów" lepiej by oddawała to, co właściwie chcemy zrobić.

Wzór na liczenie skuteczności modelu:

Czyli dla powyższego przykładu mamy 6 predykcji, z czego 4 się różnią

1 - 4/6 (0.6666) = 1/3 (0.333) lub po prostu 33% skuteczności?

Wymiarowo to np. tak wygląda:

Xtrain - (40, 100) (40 znaków różnych, każdy składa się z 100 pikseli)

Ytrain - (40,1) (klasy dla tych 40 znaków)

Xnew - (30, 100) (30 nowych znaków do sklasyfikowania)

Yreal - (30,1) (klasy dla tych 30 nowych znaków (faktycznie/realne))

Ypred - (30,1) (przewidywane klasy dla tych 30 nowych znaków)

Ogólnie w Hammingu to jest tak, że bierzemy 1 wiersz z **Xnew** i porównujemy z wszystkimi wierszami z **Xtrain** (obliczamy odległość/ XOR'ujemy wiersze)

Więc z jednego takiego porównania otrzymalibyśmy **macierz odległości** o wymiarach

(1,40) (bo 1 wiersz porównujemy z 40. wierszami z Xnew)

Np. (4 3 6 10 11 3 2) < - to są odległości dla jednej takiej iteracji [1 wiersz Xnew <-> macierz Xtrain]

I tak powtarzając 30 razy, bo Xnew zawiera 30 znaków wychodzi nam macierz odległości:

(30,40)

Czyli 1 wiersz zawiera 40 odległości.

W numpy jest taka funkcja jak **np.argsort**, która zwraca nam indeksy, wg. których podana macierz/wektor byłby posortowany.

Przykład dla wektora:

5 6 3 2 1

np.argsort zwróciłby coś w stylu

array([3, 4, 2, 1, 0]

(5 idzie na 3cią pozycję, 6 na 4tą, 3 na 2gą itd.)

W pythonie jest bodajże taki fajny feature, że można sobie tą macierzą/wektorem indeksów uporządkować inną macierz/wektor (y[w]? gdzie **y** to macierz/wektor który chcemy sobie uporządkować, **w** to ta macierz/wektor indeksów).

Detali nie pamiętam, było w zadaniu 2gim, możecie na necie doczytać, informuję tylko, że się da :D

I wg. tego sposobu tworzymy macierz o rozmiarach identycznych z **macierzą odległości** (30,40) z tą różnicą, że zawiera ona numerki klas, dla odległości posortowanych.

No i tak jak wcześniej pisałem. Z tej utworzonej macierzy klasa/odległość z każdego wiersza wybieramy **k** pierwszych wpisów, sprawdzamy dla każdego wiersza, która klasa najczęściej występuje i z tego ostatecznie tworzymy wektor **Ypred**, który zawiera nasze predykowane klasy.

Jak odczytać dane z train.pkl (nie gwarantuję, że zadziała ;D):

Wrzucamy
Import pickle as pkl

nazwa_zmiennej= pkl.load(open('train.pkl', mode='rb'))

nazwa_zmiennej[0] – wydobywamy wszystkie X
nazwa_zmiennej[1] – wydobywamy wszystkie Y odpowiadające

nazwa_zmiennej[0/1][dolnyIndeks : gornyIndeks]
lub
nazw_zmiennej [0/1][:gornyIndeks] (bierze wszystkie pozycje od 0 do gornyIndeks

Podsumowanie:

1. Wyciągamy sobie z train.pkl jakiś zbiór treningowy (np. 3k x-ów i y-ków) Xtrain i Ytrain
2. Osobno wyciągamy Xnew i Yreal, które posłużą za zbiór walidacyjny.
3. Xnew hammingujemy z Xtrain i tworzymy macierz odległości
4. Sortujemy odległości na podstawie rozmiaru rosnąco
5. Ustawiamy z Yreal klasy tak, żeby odpowiadały posortowanym pozycjom z macierzy odległości – powstaje z tego macierz klasa/odległość
6. Bierzemy z każdego wiersza macierzy klasa/odległość **k** pierwszych pozycji. Najczęściej powtarzająca się klasa to nasza predykcja.
7. Wrzucamy tą klasę do odpowiadającego wiersza Ypred
8. Porównujemy Ypred i Yreal. Liczymy liczbę pozycji na których się różnią, sumę dzielimy przez całkowity rozmiar Ypred.
9. Dla znalezienia **k** dobrego można model selection zaaplikować i wypróbować dla różnych wartości **k.**

Podziękowania dla **Mateusz Mati i Michał Karol**

Bez których nie dałoby radę tego wykminić (o ile dobrze to zrozumiałem :P)

~Daro