Podstawy sztucznej inteligencji

Laboratorium 3

Uczenie pojedynczego perceptronu

## Definicja problemu

Dany jest zbiór punktów na płaszczyźnie. Prosta o równaniu Ax + By + C = 0 dzieli ten zbiór na dwie klasy. Zaimplementuj klasyfikator oparty o pojedynczy perceptron, który otrzymując na wejściu współrzędne punktu, zaklasyfikuje go do właściwej klasy. Przebieg uczenia zaprezentuj na wykresie.

Uwagi:

1. Zbiór punktów należy wygenerować losowo, a następnie przy pomocy wybranej prostej przypisać każdemu etykietę klasy
2. Zbiór należy podzielić na podzbiory: testowy i treningowy
3. Wynik klasyfikacji podajemy procentowo, dzieląc liczbę prawidłowo zaklasyfikowanych przykładów przez wszystkie przykłady

## Rozwiązanie i opis

Generowanie zbioru i podział na podzbiory: treningowy i testowy

def generate\_two\_sets(size\_of\_base\_set): # will be split 4:1 | pass dividible by 5  
 base\_set = []  
 test\_set = []  
 rand\_weights = [round(random.uniform(-10, 10), 2), round(random.uniform(-10, 10), 2)]  
 for i in range(0, size\_of\_base\_set):  
 temp = []  
 temp.append(round(random.uniform(-10, 10), 2))  
 temp.append(round(random.uniform(-10, 10), 2))  
 if temp[0] \* rand\_weights[0] + temp[1] \* rand\_weights[1] > 0:  
 temp.append(1)  
 else:  
 temp.append(-1)  
 base\_set.append(temp)  
 for j in range(0, int(size\_of\_base\_set / 5)): #\*4  
 test\_set.append(base\_set.pop(random.randint(0, len(base\_set) - 1)))  
 return base\_set, test\_set

Funkcja ***generate\_two\_sets*** przyjmuje jeden parametr, rozmiar zbioru do wygenerowania. Przekazany argument powinien być liczbą całkowitą dodatnią podzielną przez 5 ze względu na występujący na końcu działania funkcji podział. Funkcja generuje losowo współrzędne dla punktów z zakresu (-10, 10) (float przycięty do 2 miejsc po przecinku), a następnie przypisywana jest do niego grupa (-1, 1) w oparciu o wygenerowane wcześniej wagi.

Tak przygotowany zestaw punktów rozdzielany jest następnie na 2 zbiory: treningowy oraz testowy w stosunku 4:1 i zwracany z funkcji.

Prezentacja graficzna

Do graficznego prezentowania postępu w dopasowaniu prostej przez algorytm zaimplementowana została funkcja ***generate\_plot***. Prezentuje ona punkty wraz z indykacją, do której podgrupy należą (z wykorzystaniem koloru) oraz prostą wygenerowaną o aktualne wartości wag

def generate\_plot(weights, bias, current\_generation, points\_dict):  
 plt.clf()  
 if (current\_generation == -1):  
 plt.title("verifivation")  
 else:  
 plt.title("generation {0}".format(current\_generation))  
 plt.grid(False)  
 plt.xlim(-11, 11)  
 plt.ylim(-11, 11)  
 xA = 11  
 xB = -11  
  
 if weights[1] != 0:  
 yA = (- weights[0] \* xA - bias) / weights[1]  
 yB = (- weights[0] \* xB - bias) / weights[1]  
 else:  
 xA = - bias / weights[0]  
 xB = - bias / weights[0]  
 yA = 11  
 yB = -11  
  
 plt.plot([xA, xB], [yA, yB], color='g', linestyle='-', linewidth=2)  
 x\_coords, y\_coords = get\_points(points\_dict, '-1')  
 plt.plot(x\_coords, y\_coords, 'bo')  
 x\_coords, y\_coords = get\_points(points\_dict, '1')  
 plt.plot(x\_coords, y\_coords, 'ro')  
 plt.show()  
 plt.pause(0.5)

Poniżej przykład działania funkcji



Funkcja aktywacji

def activation(output, threshold):  
 if output > threshold:  
 return 1  
 else:  
 return -1

Nauka algorytmu

data\_dictionary = {}  
for line in training\_array:  
 data\_dictionary['{0},{1}'.format(line[0], line[1])] = '{0}'.format(line[2])  
test\_dictionary = {}  
for line in test\_array:  
 test\_dictionary['{0},{1}'.format(line[0], line[1])] = '{0}'.format(line[2])  
  
  
weights = [round(random.uniform(-10, 10), 2), round(random.uniform(-10, 10), 2)]  
bias = 1  
threshold = 0  
learning\_rate = 0.05  
max\_iterations = 1000  
plt.ion()  
  
for k in range(1, max\_iterations):  
 hits = 0  
 print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_generation{0}\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_".format(k))  
 for i in range(0, len(training\_array)):  
 sum = 0  
 for j in range(0, len(training\_array[i]) - 1):  
 sum += training\_array[i][j] \* weights[j]  
 output = bias + sum  
 y = activation(output,threshold)  
 ### weights correction ###  
 if y == training\_array[i][2]:  
 hits += 1  
 else:  
 for j in range(0, len(weights)):  
 weights[j] = weights[j] + (learning\_rate \* training\_array[i][2] \* training\_array[i][j])  
 bias = bias + learning\_rate \* training\_array[i][2]  
 generate\_plot(weights, bias, k, data\_dictionary)  
 if hits == len(training\_array):  
 print("-------------------------------------------------------------")  
 print("ALgoritm learned within {0} iterations".format(k))  
 break  
print(weights)  
print("DONE")

Dla wygenerowanych losowo punktów następuje przejście przez kolejne epoki do momentu uzyskania wag odpowiadających prostej, dla której wszystkie punkty klasyfikowane są prawidłowo, lub osiągnięcia maksymalnej liczby epok. Na koniec każdej epoki generowany jest wykres przedstawiający aktualną prostą na tle punktów ze zbioru treningowego.

Weryfikacja

generate\_plot(weights, bias, -1, test\_dictionary)  
all\_points = len(test\_array)  
correctly\_classified = 0  
for each in test\_array:  
 output = each[0] \* weights[0] + each[1] \* weights[1]  
 if activation(output, threshold) == each[2]:  
 correctly\_classified +=1  
print("{0}% of poits were classified correctly".format(round(correctly\_classified/all\_points\*100),2))

Wygenerowany zbiór testowy poddawany jest klasyfikacji w oparciu o wagi uzyskane w wyniku nauki. Procentowy udział poprawnie zaklasyfikowanych punktów w całym zbiorze testowym drukowany jest następnie na ekran.

Przykładowe wyniki

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_generation1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_generation2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_generation3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-------------------------------------------------------------

Algorithm learned within 3 iterations

[5.5745000000000005, 4.42]

DONE

100% of points were classified correctly

   

Porównanie skuteczności klasyfikacji dla różnych wielkości zbioru do nauki

**Zbiór bazowy: 1010 | Zbiór do nauki: 10 | Zbiór testowy : 1000**

Loop0: 92.7% of points were classified correctly

Loop1: 99.2% of points were classified correctly

Loop2: 96.2% of points were classified correctly

Loop3: 99.7% of points were classified correctly

Loop4: 93.5% of points were classified correctly

Loop5: 98.5% of points were classified correctly

Loop6: 91.2% of points were classified correctly

Loop7: 98.7% of points were classified correctly

Loop8: 84.5% of points were classified correctly

Loop9: 95.1% of points were classified correctly

**Zbiór bazowy: 1100 | Zbiór do nauki: 100 | Zbiór testowy : 1000**

Loop0: 99.5% of points were classified correctly

Loop1: 97.5% of points were classified correctly

Loop2: 98.5% of points were classified correctly

Loop3: 99.7% of points were classified correctly

Loop4: 99.8% of points were classified correctly

Loop5: 99.9% of points were classified correctly

Loop6: 99.9% of points were classified correctly

Loop7: 98.6% of points were classified correctly

Loop8: 99.2% of points were classified correctly

Loop9: 99.5% of points were classified correctly

**Zbiór bazowy: 2000 | Zbiór do nauki: 1000 | Zbiór testowy : 1000**

Loop0: 100.0% of points were classified correctly

Loop1: 100.0% of points were classified correctly

Loop2: 99.8% of points were classified correctly

Loop3: 99.8% of points were classified correctly

Loop4: 99.8% of points were classified correctly

Loop5: 99.9% of points were classified correctly

Loop6: 99.7% of points were classified correctly

Loop7: 100.0% of points were classified correctly

Loop8: 99.9% of points were classified correctly

Loop9: 100.0% of points were classified correctly

## Wnioski i obserwacje

Skuteczność działania algorytmów uczących bardzo mocno zależy od zbioru danych treningowych oparciu, o które odbyła się nauka. Podobnie jak w przypadku nauki u ludzi, dość łatwo można wpaść pułapkę zbyt podobnych danych wejściowych. W przypadku użycia zbyt podobnych, lub posiadających takie same cechy danych wejściowych uzyskać możemy fałszywe przekonanie o nauczeniu prawidłowego sposobu rozwiązywania problemu. Możliwie duże urozmaicenie tych danych pozwala na uzyskanie lepszych wyników (w przypadku implementowanego rozwiązania – użycie losowości przy generowaniu danych).

Równie duży wpływ na skuteczność uczenia ma wielkość zbioru do nauki. Jak można zauważyć na przedstawionych powyżej wynikach, wzrost liczności zbioru do nauki o rząd wielkości, nawet przy tak prostym problemie skutkował znacznym poprawieniem skuteczności klasyfikacji.