Paweł Jędrzejczyk 224819

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,

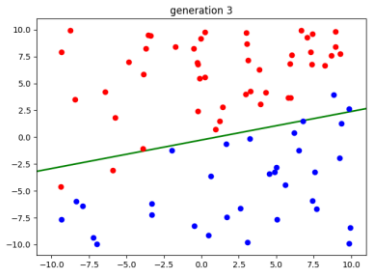
do której podgrupy należą (z wykorzystaniem koloru) oraz prostą wygenerowaną o aktualne wartości wag

```
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.

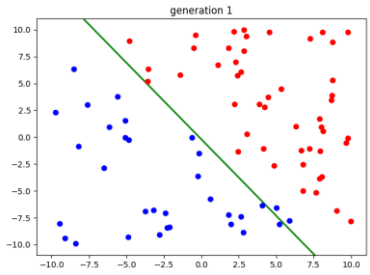
Weryfikacja

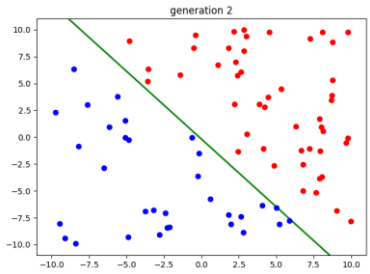
```
generate_plot(weights, bias, -1, test_dictionary)
all_points = len(test_array)
correctly_classified = 0
for each in test_array:
    output = each[0] * weights[0] + each[1] * weights[1]
    if activation(output, threshold) == each[2]:
        correctly_classified +=1
print("{0}% of poits were classified
correctly".format(round(correctly_classified/all_points*100),2))
```

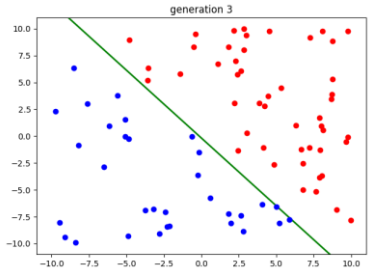
Wygenerowany zbiór testowy poddawany jest klasyfikacji w oparciu o wagi uzyskane w wyniku nauki. Procentowy udział poprawnie zaklasyfikowanych punktów w całym zbiorze testowym drukowany jest następnie na ekran.

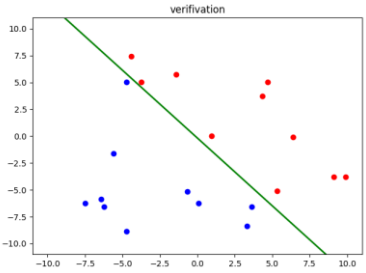
Przykładowe wyniki

```
_________________generation1_________________
_________________generation2_________________
_________________generation3_________________
-------------------------------------------------------------
Algorithm learned within 3 iterations
[5.5745000000000005, 4.42]
DONE
100% of points were classified correctly
```


generation 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


generation 2
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


generation 3
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



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.