exc1

August 23, 2018

```
In [1]: import numpy as np
        import pandas as pd

        import plotly

        def show_verbose(do_it, str_data):
            if do_it:
                print(str_data)
            else:
                pass
            return None
```

1 EXERCÍCIO 1

1. Implementar e treinar o modelo Adaline para reconhecer os símbolos **A e A invertida**).
2. Faça uma representação matricial de "-1" e "+1" para desenhar esses símbolos graficamente, e crie vários exemplos de treinamento e teste, inserindo ruídos arbitrariamente.
3. Por exemplo, para representar graficamente o símbolo A invertido em uma matriz 5x5, uma possibilidade seria:

	A	B	C	D	E
a	**+1**	*-1*	*-1*	*-1*	**+1**
b	**+1**	*-1*	*-1*	*-1*	**+1**
c	*-1*	**+1**	**+1**	**+1**	*-1*
d	*-1*	**+1**	*-1*	**+1**	*-1*
e	*-1*	*-1*	**+1**	*-1*	*-1*

4. Cada valor binário contido na representação (i.e., cada "-1" ou "+1") é uma entrada de um exemplo.
5. Lembre-se de que todos os exemplos devem ser rotulados, com "-1" para A e "+1" para A invertido (ou vice-versa).
6. Crie no mínimo 6 exemplos com cada rótulo (total: 12 exemplos).

7. Utilize linguagem de programação Python.

8. Elabore um **relatório**, de 1 a 2 páginas, descrevendo o que foi feito e mostrando os resultados nos conjuntos de treinamento e teste.

- Deverão ser postados no escaninho do Tidia, em um *único arquivo compactado, com extensão .zip ou .rar, intitulado* **"_exercicio1.zip"ou "_exercicio1.rar"**:
  - relatório
  - código-fonte
  - exemplos criados para uso como entradas.

1.1 1. Modelo Adaline

The ***Adaline*** model is an anagram name for the perceptro who uses the delta rule (*gradient*)

```
In [2]: def perceptron_core(input_data, output_data, amount_of_cases = None,
                            teacher=0.5, epochs=10000, weigth_max = 1e0,
                            verbose=True, verbose_ratio=1e-2):

            if amount_of_cases is None:
                amount_of_cases = input_data.shape[0]
            else:
                pass

            case_size = input_data.shape[1]

            weigth_data = np.random.rand(case_size) * weigth_max
            theta = np.random.randint(1<<8)

            f_perp = None
            old_diff = None
            for epoch in range(epochs):
                for case_id in range(amount_of_cases):
                    f_perp = np.dot(input_data[case_id], weigth_data.T)
                    f_perp += theta

                    diff_perp = output_data[case_id] - f_perp

                    weigth_data = weigth_data + ((teacher * diff_perp) *  input_data[case_id])
                    theta = theta + (teacher * diff_perp)

                    if verbose:
                        if  epoch % int(epochs * verbose_ratio) == 0:
                            print("Obtain : {:f}\tExpected : {:f}".format(f_perp, output_data[ca

                if verbose:
                    if epoch % int(epochs * verbose_ratio) == 0:
                        print("Epoch {:d}".format(epoch))
```

```
            if verbose:
                print("Epoch {:d}".format(epoch))
                print("Obtain : {:f}\tExpected : {:f}".format(f_perp, output_data[case_id]))

            return [weigth_data, theta]
```

```
In [3]: def perceptron_predict(test_cases, test_labels, weigth_data, theta):
            output = []

            for id_case in range(test_cases.shape[0]):
                predicted_output = np.dot(weigth_data, test_cases[id_case].T) + theta
                output.append([predicted_output, test_labels[id_case]])

            return np.matrix(output)
```

```
In [4]: dataset = pd.read_csv("./dataset.csv")
```

```
In [5]: dataset_matrix = dataset.as_matrix()
        samples = dataset_matrix[:,1:]
        samples_n = samples[:samples.shape[0]//2]
        samples_i = samples[samples.shape[0]//2:]
```

2 Training Set

```
In [6]: def sub_sampler(data_samples, ratio, verbose=False):
            top_id = int(samples_n.shape[0] * ratio)

            show_verbose(verbose, top_id)

            rows_id = np.arange(data_samples.shape[0])
            sub_sample_ids_train = np.random.choice(rows_id, top_id, replace=False)

            show_verbose(verbose, sub_sample_ids_train)

            sub_sample_ids_test = np.setdiff1d(rows_id, sub_sample_ids_train)

            show_verbose(verbose, sub_sample_ids_test)

            return [data_samples[sub_sample_ids_train, :],data_samples[sub_sample_ids_test, :]]
```

We have 30 samples (labeled data) , we are gona use the 70% for traing and the rest (30%) for testing

```
In [7]: samples_n_mtx = sub_sampler(samples_n, 0.70)
        samples_i_mtx = sub_sampler(samples_i, 0.70)
```

```
In [8]: train_set = np.concatenate((samples_n_mtx[0], samples_i_mtx[0]), axis=0)
        test_set = np.concatenate((samples_n_mtx[1], samples_i_mtx[1]), axis=0)
```

```
In [9]: [trained_weight, theta] = perceptron_core(train_set[:, 1:], train_set[:,0],
                                                  teacher=0.02,epochs=int(1.5e4),
                                                  verbose=True, verbose_ratio=8e-1)
```

```
Obtain : 119.200566        Expected : 1.000000
Obtain : 68.886611         Expected : 1.000000
Obtain : 39.409884         Expected : 1.000000
Obtain : 22.005491         Expected : 1.000000
Obtain : 12.771121         Expected : 1.000000
Obtain : 6.732755         Expected : 1.000000
Obtain : 4.899538         Expected : 1.000000
Obtain : 5.197386         Expected : 1.000000
Obtain : 1.685970         Expected : 1.000000
Obtain : -9.842781         Expected : 1.000000
Obtain : 7.569348         Expected : 1.000000
Obtain : 1.397708         Expected : 1.000000
Obtain : 2.238933         Expected : 1.000000
Obtain : -0.211663         Expected : 1.000000
Obtain : 1.178900         Expected : 1.000000
Obtain : 2.192436         Expected : 1.000000
Obtain : -8.377314         Expected : 1.000000
Obtain : 122.628165        Expected : -1.000000
Obtain : 91.344924         Expected : -1.000000
Obtain : 50.387733         Expected : -1.000000
Obtain : 5.717568         Expected : -1.000000
Obtain : 5.370333         Expected : -1.000000
Obtain : 21.114884         Expected : -1.000000
Obtain : 7.908183         Expected : -1.000000
Obtain : 5.695544         Expected : -1.000000
Obtain : 4.209851         Expected : -1.000000
Obtain : -19.099433        Expected : -1.000000
Obtain : -9.687728         Expected : -1.000000
Obtain : 15.087520         Expected : -1.000000
Obtain : 1.172722         Expected : -1.000000
Obtain : -13.911494        Expected : -1.000000
Obtain : 12.048156         Expected : -1.000000
Obtain : -13.501098        Expected : -1.000000
Obtain : 10.692925         Expected : -1.000000
Epoch 0
Obtain : 1.000000         Expected : 1.000000
Obtain : 1.000000         Expected : 1.000000
Obtain : 1.000000         Expected : 1.000000
Obtain : 1.000000         Expected : 1.000000
Obtain : 1.000000         Expected : 1.000000
Obtain : 1.000000         Expected : 1.000000
Obtain : 1.000000         Expected : 1.000000
Obtain : 1.000000         Expected : 1.000000
Obtain : 1.000000         Expected : 1.000000
```

```
Obtain : 1.000000        Expected : 1.000000
Obtain : 1.000000        Expected : 1.000000
Obtain : 1.000000        Expected : 1.000000
Obtain : 1.000000        Expected : 1.000000
Obtain : 1.000000        Expected : 1.000000
Obtain : 1.000000        Expected : 1.000000
Obtain : 1.000000        Expected : 1.000000
Obtain : 1.000000        Expected : 1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Obtain : -1.000000        Expected : -1.000000
Epoch 12000
Epoch 14999
Obtain : -1.000000        Expected : -1.000000
```

```
In [10]: dummy_predict = perceptron_predict(test_set[:, 1:], test_set[:,0], trained_weight, thet
```

```
In [11]: dummy_final = ((dummy_predict < 0).astype(int) * -1) + (dummy_predict > 0).astype(int)
         differ_dummy_final = (dummy_final[:, 0] != dummy_final[:, 1]).astype(int)
```

3 Results

3.1 Raw Results

```
In [12]: pd.DataFrame(np.concatenate((dummy_predict, differ_dummy_final), axis=1),
                      columns=['Predicted', 'Expected', 'Different'])
```

Out[12]:

	Predicted	Expected	Different
0	0.727216	1.0	0.0
1	1.000000	1.0	0.0
2	1.000000	1.0	0.0
3	1.000000	1.0	0.0
4	1.000000	1.0	0.0

```
5    1.000000       1.0        0.0
6    1.000000       1.0        0.0
7    1.000000       1.0        0.0
8   -1.000000      -1.0        0.0
9   -1.000000      -1.0        0.0
10  -0.727216      -1.0        0.0
11  -0.193633      -1.0        0.0
12  -1.000000      -1.0        0.0
13  -1.000000      -1.0        0.0
14  -1.000000      -1.0        0.0
15  -1.000000      -1.0        0.0
```

3.2 Formated Results

```
In [13]:  pd.DataFrame(np.concatenate((dummy_final, differ_dummy_final), axis=1),
                       columns=['Predicted', 'Expected', 'Different'])
```

Out[13]:

	Predicted	Expected	Different
0	1	1	0
1	1	1	0
2	1	1	0
3	1	1	0
4	1	1	0
5	1	1	0
6	1	1	0
7	1	1	0
8	-1	-1	0
9	-1	-1	0
10	-1	-1	0
11	-1	-1	0
12	-1	-1	0
13	-1	-1	0
14	-1	-1	0
15	-1	-1	0