Dual Perceptron

Il Perceptron è un metodo di apprendimento supervisionato per la creazione di classificatori lineari ideato da Frank Rosenblatt nel 1956. Esso è un metodo di apprendimento lineare in grado di migliorare la sua precisione dai propri errori. Ovvero esso consiste nel determinare un iperpiano di separazione dei campioni di addestramento di un dato dataset (fit del dataset). Alla fine di questa fase di addestramento, l'algoritmo utilizza gli errori che compie un aggiornamento dell'iperpiano fino a quando non ci saranno più errori. Quando i dati di addestramento non sono separabili, l'algoritmo potrebbe non convergere ed essere interrotto arbitrariamente dopo un cero numero di iterazioni.

La funzione di decisione lineare è: if $(\vec{w}^T \vec{X} + b) > 0$ then positive else negative

Assumiamo di avere un set di training con M esempi $S = \{\vec{X}_m, y_m\}$, in cui y assume valore +1 e -1 in caso di rilevamenti positivi o negativi rispettivamente. Il classificatore è definito dal vettore dei pesi W, e dal bias b.

Esso classifica in maniera corretta l'esempio $(\vec{X}_m, y_m)$ se

$$y_m(\vec{w}^T \vec{X}_m + b) > 0$$

L'algoritmo usa la regola di aggiornamento:

$$\text{if } y_m(\vec{w}_i^T \vec{X}_m + b) \leq 0 \text{ then } \vec{w}_{i+1}^T \leftarrow \vec{w}_i^T + \eta\, y_m \vec{X}_m$$

L'algoritmo completo:

Given a linearly separable training set S and learning rate $\eta \in \mathbb{R}^+$
$\mathbf{w}_0 \leftarrow \mathbf{0}$; $b_0 \leftarrow 0$; $k \leftarrow 0$
$R \leftarrow \max_{1 \leq i \leq \ell} \|\mathbf{x}_i\|$
repeat
  for $i = 1$ to ℓ
    if $y_i(\langle \mathbf{w}_k \cdot \mathbf{x}_i \rangle + b_k) \leq 0$ then
      $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \eta y_i \mathbf{x}_i$
      $b_{k+1} \leftarrow b_k + \eta y_i R^2$
      $k \leftarrow k + 1$
    end if
  end for
until no mistakes made within the *for* loop
return $(\mathbf{w}_k, b_k)$ where k is the number of mistakes

(Cristianini Shawe - Perceptron algorithm primal form)

Perceptron forma duale

La forma duale dell'algoritmo Perceptron permette di applicare l'algoritmo anche a problemi non linearmente separabili, e grazie alla funzione kernel permette di trovare un iperpiano che separi i dati in modo più efficiente. In questo caso entrano in gioco un vettore alfa, inizialmente nullo, e ogni volta che l'algoritmo sbaglia a classificare i vari elementi y[i] verrà incrementato di uno il corrispettivo alfa (alfa[i]).

La nuova funzione di classificazione sarà:

$$f(\vec{X}) = \vec{w}^T \vec{X} + b = \sum_{m=1}^{M} a_m \, y_m \left\langle \vec{X}_m, \vec{X} \right\rangle + b$$

L'algoritmo completo sarà il seguente

Given training set S
$\boldsymbol{\alpha} \leftarrow \mathbf{0}$; $b \leftarrow 0$
$R \leftarrow \max_{1 \le i \le \ell} \|\mathbf{x}_i\|$
repeat
  for $i = 1$ to ℓ
    if $y_i \left(\sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x}_i \rangle + b \right) \le 0$ then
      $\alpha_i \leftarrow \alpha_i + 1$
      $b \leftarrow b + y_i R^2$
    end if
  end for
until no mistakes made within the *for* loop
return $(\boldsymbol{\alpha}, b)$ to define function $h(\mathbf{x})$ of equation (2.1)

(Cristianini Shawe – Perceptron Algorithm Dual form)

I dataset utilizzati nella mia implementazione dell'algoritmo sono :

- Biodegradation Data Set
- Qsar Oral Toxicity
- Red Wine quality

I quali sono stati caricati attraverso il metodo read_csv() della libreria pandas

```
def dataset_(dcName, delimiter):

    if dcName == 'qsar_oral_toxicity':
        # delimiter ';'
        df = pd.read_csv(r'dataset\qsar_oral_toxicity.csv', delimiter)
        cleanup_nums = {"positive": 1, "negative": -1}
        df.replace(cleanup_nums, inplace=True)
        df = shuffle(df)
        return df
```

I dataset possono essere inseriti nella creazione dell'istanza della classe DualPerceptron, inserendoli come:

'biodeg', 'qsar_oral_toxicity', 'winequality-red'.

I singoli dataset sono stati "splittati" in tre set distinti: train_set, validation_set, test_set.

Train_set e validation_set sono usati durante la fase di addestramento (fit()), il primo fornisce i valori su cui addestrare l'algoritmo e su cui verrà costruito l'iperpiano, il secondo viene usato sempre in questa fase per calcolare il tasso di accuratezza dell'algoritmo ad ogni epoca, e nel caso in cui il valore di accuratezza inizi a scendere rispetto alle prime epoche, viene bloccato, perché significa che sta iniziando ad "imparare troppo bene il train_set".

Le funzioni kernel utilizzate sono:

- Linear kernel
- Polinomial kernel
- RBF kernel

```
def kernel_(x, y, kernel):
    op = 0
    if kernel == 'linear_kernel':
        op = np.dot(x, y)
    elif kernel == 'poly_kernel':
        op = (np.dot(x, y)+1) ** 5
    elif kernel == 'RBF_kernel':
        gamma = 1.0
        op = np.exp(-gamma * np.linalg.norm(x - y) ** 2)
    return op
```

Durante la fase di addestramento è necessario il calcolo della marice di Gram:

```
### gramMatrix
print('gram matrix init')
x_count = len(self.X_train)
gramM = np.zeros((x_count, x_count))  ### np.zeros(dimX, dimX)
for i in range(len(self.X_train)):
    for j in range(len(self.X_train)):
        gramM[i][j] = kernel_(self.X_train[i], self.X_train[j], self.kernel)
print("gram matrix finish")
self.gramMat = gramM
###
```

I risultati che si ottengono con i tre kernel sono molto diversi, dipendono da come vengono “mescolati i dati” dei datatset attraverso il metodo shuffle()