Perceptron vs. Voted Perceptron

Emanuele Vivoli

February 2018

1 Accenni Teorici

Nell'apprendimento automatico, il Perceptron è un tipo di classificatore binario che mappa i suoi ingressi $\mathbf{x}$ (un vettore di tipo reale) in un valore di output $f(\mathbf{x})$ (uno scalare di tipo reale) calcolato con:

$f(x) = \chi(\langle \mathbf{w}, \mathbf{x} \rangle + b)$

dove $\mathbf{w}$ è un vettore di pesi con valori reali, l'operatore $\langle \cdot, \cdot \rangle$ è il prodotto scalare (che calcola una somma pesata degli input), b è il 'bias', un termine costante che non dipende da alcun valore in input e $\chi\,(y)$ è la funzione di output. La nostra scelta per la funzione $\chi(y)$ è:

$\chi(y) = sign(y)$

Dunque la l'algoritmo corrisponde a un classificatore binario (l'output può assumere solamente i valori $+1$ e -1).

Una variante del Perceptron è chiamata Voted Perceptron ed utilizza una formula che tiene traccia di tutti gli iperpiani che sono stati calcolati per separare linearmente il dataset, secondo la formula di valutazione di un elemento:

$f(x) = sign(s)$;

$s = \sum_{i=1}^{k} ci * sign(\langle \mathbf{wi}, \mathbf{x} \rangle)$

Associa infatti ad ogni iperpiano un peso, in base a quanti elementi è "resistito" l'iperpiano. Confronta quindi ogni elemento del Test con ogni iperpiano generato ed attribuisce una classe in base a come è stato maggiormente classificato.

2 Implementazione in Python

Nel file "perceptron.py" è implementata la classe Perceptron, mentre in "voted_perceptron.py" è implementata la classe Voted Perceptron. I metodi fondamentali sono $training(\mathbf{X}, \mathbf{Y})$ e $predict(\mathbf{xi})$, con $\mathbf{X}$ array di elementi $\mathbf{xi}$ (punto in $R^{len(\mathbf{xi})}$ dimensioni) e $\mathbf{Y}$ array che ne identifica la classe.

2.1 Perceptron algothm

```
def training(self, X, Y):
    self.w = np.zeros(len(X[0]))
    k = 0
    i = 0
    max_ = self.max_norm(X)
    while True:
        n_err = 0
        for xi, yi in zip(X, Y):
            if yi * self.predict(xi) < 0:
                self.w[1:] = [x + self.eta * (y * yi)
                              for x, y in zip(self.w[1:], xi)]
                self.w[0] += self.eta * yi * math.pow(max_, 2)
                k += 1
                n_err += 1
        i += 1
        if n_err == 0 or i > self.epochs:
            break
    return self.w, i - 1, k

def net_input(self, xi):
    return np.dot(xi, self.w)

def predict(self, xi):
    return 1 if self.net_input(xi) >= 0.0 else -1
```

2.2 Voted Perceptron algothm

```
def training(self, X, Y):
    self.k = 0
    self.W = [np.zeros(len(X[0]))]
    self.c = [0]
    self.w = np.zeros(len(X[0]))
    i = 0
    while True:
        n_err = 0
        for xi, yi in zip(X, Y):
            if yi * (1 if np.dot(xi, self.w) >= 0 else -1) >= 0:
                self.c[self.k] += 1
            else:
                self.W.append([(w + yi * x)
                               for w, x in zip(self.w, xi)])
                self.w = self.W[self.k + 1]
                self.c.append(1)
                self.k += 1
```

```
                n_err += 1

        i += 1
        if n_err == 0 or i > self.epochs:
            break
    return self.w, i - 1, len(self.W)


def net_input(self, xi):
    s = [(1 if np.dot(xi, self.W[i]) >= 0 else -1) * self.c[i]
         for i in np.arange(0, self.k + 1, 1)]
    return sum(s)


def predict(self, xi):
    return 1 if self.net_input(xi) >= 0 else -1
```

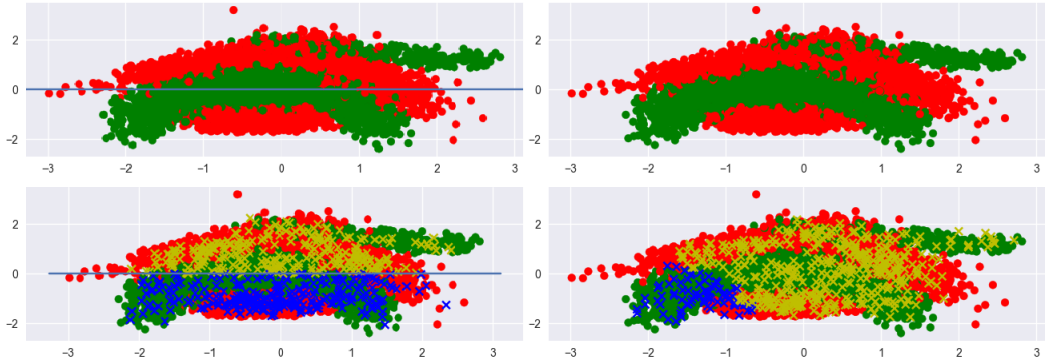
3 Risultati sperimentali

Viene richiesto dall'esercizio di utilizzare tre datasets dal sito mldata e tramite la cross validation allenare gli algoritmi con il Train e testarli mediante il Test (è stata utilizzata la cross validation tramite sklearn.train_set_split con un indice 0.1 che corrisponde ad una 10fold cross validation).

I tre datasets sono BANANA, MUSHROOM e PAGEBLOCKS.

3.0.1 Banana

BANANA è un dataset di punti in uno spazio bidimensionale ma non è linearmente separabile, dunque gli errori che ci aspettiamo possono essere della forma più disparata. Per questo caso non ha molto senso fare un confronto tra i due algoritmi e le loro prestazioni, ma potrebbe essere interessante vedere come i due algoritmi classificano i punti in base alla ultima retta trovata (sarebbe un iperpiano in spazi di dimensione maggiore di 2) e in base alle rette precedentemente costruite.



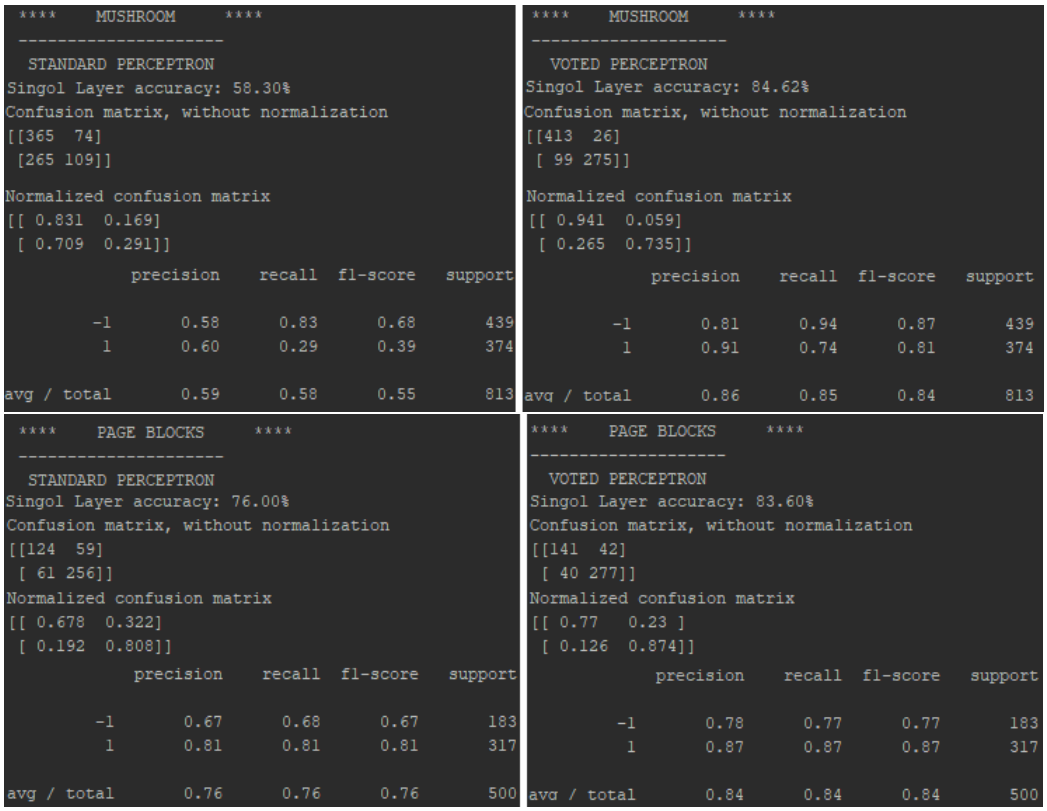


Figure 1: Mushroom e Pageblocks output

3.0.2 Mushroom e Pageblocks

MUSHROOM e PAGEBLOCKS invece ci permettono di vedere come l'algoritmo Voted Perceptron abbia una precisione migliore nel classificare gli elementi di Test, rispetto al Perceptron non votato. Questi database sono infatti linearmente separabile con alta probabilità).

Per informazioni sulla lettura dell'output si rimanda il lettore a consultare il file README del progetto su github.