Implementazione in Python di un modello di apprendimento lineare

In questo documento viene brevemente descritto *un algoritmo* (semplificato) per l'addestramento di un modello lineare.
Nello specifico, è presentata una implementazione in Python del solo algoritmo di training, fornita come strumento basilare e a solo scopo illustrativo e didattico (come complemento per aiutare la comprensione dell'algoritmo sulle slide del corso).
L'invocazione e l'uso appropriato è demandato a funzioni ausiliari e alla loro invocazione (vedi demo).

L'uso del codice Python è quindi da intendere con alcune avvertenze (warning). A titolo di esempio, per il modello lineare:

- il controllo di fine algoritmo non ha un criterio di uscita significativo (vi è qui semplicemnte un ciclo for sul numero di epoche, fissate da utente, con default uguale a 100);
- non vi sono strumenti per il plot della learning curve (errore ad ogni epoca) in genere utili per la gestione e analisi di tool di ML;
- le misure di performance sono limiate al Mean Square Error (MSE) e all'accuracy (per classificazone): vedi demo;
- la gestione della cross validation è implementata in modo primitivo: vedi demo;
- non ci sono ottimizzazioni nell'uso delle strutture dati, nell'uso delle matrici, ecc.
- l'uso degli indici per scegliere le variabili di input o di target è una possbilita tra le molte.

Il codice relativo all'implementazione dell'algoritmo di apprendimento per un modello lineare è riportato in Tabella 1.

Saranno forniti in file separati il codice Python, i data-set e la demo (guida/documentazione).
L'uso comunque è rimandato a dopo la lezione sulla Validation

```
def LinearLearner(dataset, learning_rate=0.01, epochs=100):
    """Addestra un modello lineare.
       In input richiede:
        dataset: oggetto della classe DataSet che contiene gli esempi
                 per il training e le funzioni per la gestione del data set
        learning_rate: eta a lezione (default 0.01)
        epochs: numero massimo di epoche di training (default 100)
       In output restituisce una lista contenente (nell'ordine):
        il predittore (per uso funzione predict, uso post training)
        i parametri addestrati (w)
        l'errore quadratico medio (MSE) di training al variare delle epoche
    """
    idx_i = dataset.inputs # indici delle variabili di input
    #nota: idx_i e' una lista degli indici nei dati di ogni esempio
    #   come [0,1,2,3], o [2,5,7]
    idx_t = dataset.target # indice della variabile usata per target
                           #(variabile target y a lezione)
    examples = dataset.examples #lista di tutti gli esempi
    #nota: ogni esempio corrisponde a una lista contenente sia le variabili
    # di input sia il target
    num_examples = len(examples) # numero di esempi nel dataset (l a lezione)
    input_dim = len(idx_i) # dimensione dell'input (n a lezione)
    #inizializza i pesi del modello in modo casuale
    #da una distribuzione uniforme tra -0.5 e 0.5
    w = [random.uniform(-0.5, 0.5) for _ in range(len(idx_i) + 1)]

    trainingError = [] # inizializza la lista che conterra' l'MSE ad ogni epoca di tr.
    for epoch in range(epochs):
        err = []
        inputs = [] #conterra' l'input di ogni esempio nel dataset
        #considera tutti gli esempi, prepara il calcolo gradiente
        for example in examples:
            x = [1] + [example[i] for i in idx_i] #input (lista locale dei
                                                  #valori delle variabili di input)
                                                  #e aggiunge l'input bias unitario
            inputs.append(x) #aggiunge l'esempio attuale alla lista
            out = dotproduct(w, x) #output del modello
            t = example[idx_t] #target (attenzione: fu y a lezione)
            err.append(t - out)  #ossia (y-xw) a lezione
        for i in range(len(w)): #calcolo il DeltaW (passo 2 alg. a lezione)
            delta_wi = 0
            for p in range(num_examples):
                x_pi = inputs[p][i] #componente i del pattern p
                delta_wi += err[p] * x_pi
            #stiamo tenendo la costante 2 come a lezione,
            #ma dividendo per il numero di esempi (l) , ossia una LMS
            delta_wi = 2 * (delta_wi /num_examples)
            w[i] = w[i] + learning_rate * delta_wi #aggiorna il peso w (passo 3 a lez.)
        # calcola l'MSE per questa epoca e lo aggiunge alla lista
        trainingError.append(sum(e*e for e in err)/len(err))

    def predict(example):
        # funzione usata per calcolare l'output del modello
        # per l'esempio specificato in input
        x = [1] + example
        return dotproduct(w, x)

    return predict, w, trainingError
```

Tabella 1 Addestramento del modello lineare.