BloomFilter Speedup on setup and filter: Parallel Programming for Machine Learning

Lorenzo Baiardi, Thomas Del Moro

GG MM AAAA

1 Introduzione

Il progetto consiste nella parallelizzazione di un algoritmo di Machine Learning, il BloomFilter, utilizzando il linguaggio di programmazione Python e la libreria Joblib. Vogliamo analizzare lo speedup ottenuto parallelizzando il BloomFilter su due principali funzioni: setup e filter.

2 Analisi del problema

Il BloomFilter è una struttura dati probabilistica che permette di verificare se un elemento appartiene ad un insieme. In questo progetto utilizziamo il bloom filter per verificare se un email è di spam o meno, dato un insieme di email su cui effettuare il training. Per la inizializzazione del bloom filter è necessario fornire la probabilità di falsi positivi e la dimensione dell'insieme su cui effettuare il training. In base al valore della probabilità di falsi positivi e alla dimensione dell'insieme, il bloom filter calcola il numero di funzioni hash da utilizzare e la dimensione del vettore di bit che permetterà di memorizzare i risultati delle funzioni hash. La formula per il calcolo del numero di funzioni hash è la seguente:

$$k = \frac{m}{n} \ln 2 \tag{1}$$

Dove m è la dimensione del vettore di bit e n è la dimensione dell'insieme. La formula per il calcolo della dimensione del vettore di bit è la seguente:

$$m = -\frac{n \ln p}{(\ln 2)^2} \tag{2}$$

Una volta passato il dataset di training al bloom filter, questo calcola per ogni email le k funzioni hash e setta a 1 i bit corrispondenti alle posizioni calcolate. Per verificare se un email è di spam o meno, il bloom filter calcola le k funzioni hash e verifica se i bit corrispondenti alle posizioni calcolate sono settati a 1.

3 Parallelizzazione

3.1 Joblib

Joblib è una libreria Python che permette di parallelizzare funzioni e cicli for. La funzione Parallel prende in input il numero di processori da

utilizzare e la funzione da parallelizzare. In questo elaborato abbiamo utilizzato la funzione Parallel per parallelizzare la funzione setup e la funzione filter del bloom filter. Abbiamo poi successivamente elaborato quanto tempo impiegano le funzioni setup e filter a seconda del numero di processori utilizzati rispetto al tempo impiegato nella sua versione sequenziale.

```
def par_filter_all(self, items, n_threads):
    # Split items in chunks
    chunks = np.array_split(items, n_threads)

    # Start parallel setup
    start = time.time()
    results = Parallel(n_jobs=n_threads)(delayed(self.seq_filter_all)(chunk) for chunk in chunks)
    end_time = time.time() - start

    # Sum errors
    t, errs = zip(*results)
    errors = sum(errs)
    return end_time, errors
```

```
def filter(self, item):
    for i in range(self.n_hashes):
        index = mmh3.hash(item, i) % self.size
        if not self.bitarray[index]:
            return False
    return True
```

```
def par_setup(self, items, n_threads, chunks=None):
    self.initialize(items)

    # Split items in chunks
    chunks = np.array_split(items, chunks if chunks else n_threads)

    # Start parallel setup
    start = time.time()
    Parallel(n_jobs=n_threads)(delayed(self.add)(chunk) for chunk in chunks)
    return time.time() - start
```

```python
def add(self, items):
    for item in items:
        for i in range(self.n_hashes):
            index = mmh3.hash(item, i) % self.size
            self.bitarray[index] = True
```

3.2 Omp

Omp è una libreria C che permette di parallelizzare funzioni e cicli for. La funzione omp parallel prende in input il numero di processori da utilizzare e la funzione da parallelizzare. In questo elaborato abbiamo utilizzato la funzione omp parallel per parallelizzare la funzione setup e la funzione filter del bloom filter. Abbiamo poi successivamente elaborato quanto tempo impiegano le funzioni setup e filter a seconda del numero di processori utilizzati rispetto al tempo impiegato nella sua versione sequenziale.

```python
def par_filter_all(self, items, n_threads):
    # Split items in chunks
    chunks = np.array_split(items, n_threads)

    # Start parallel setup
    start = time.time()
    results = Parallel(n_jobs=n_threads)(delayed(self.seq_filter_all)(chunk) for chunk in chunks)
    end_time = time.time() - start

    # Sum errors
    t, errs = zip(*results)
    errors = sum(errs)
    return end_time, errors
```

4 Tests

I test sono stati effettuati su un dataset fino a 100000 email, con una probabilità di falsi positivi pari a 0.01. Mentre la probabilità di falsi positivi è rimasta costante, la dimensione dell'insieme è stata variata da 1000 a 100000 email per verificare come varia lo speedup al variare della dimensione dell'insieme. Ci aspettiamo che per i primi test, con un numero di email minore, lo speedup sia inferiore rispetto alla versione sequenziale dovuto alla creazione dell'ambiente che permette la parallelizzazione dei processi. Con l'aumento invece delle email, ci aspettiamo che lo speedup aumenti fino ad un certo punto.

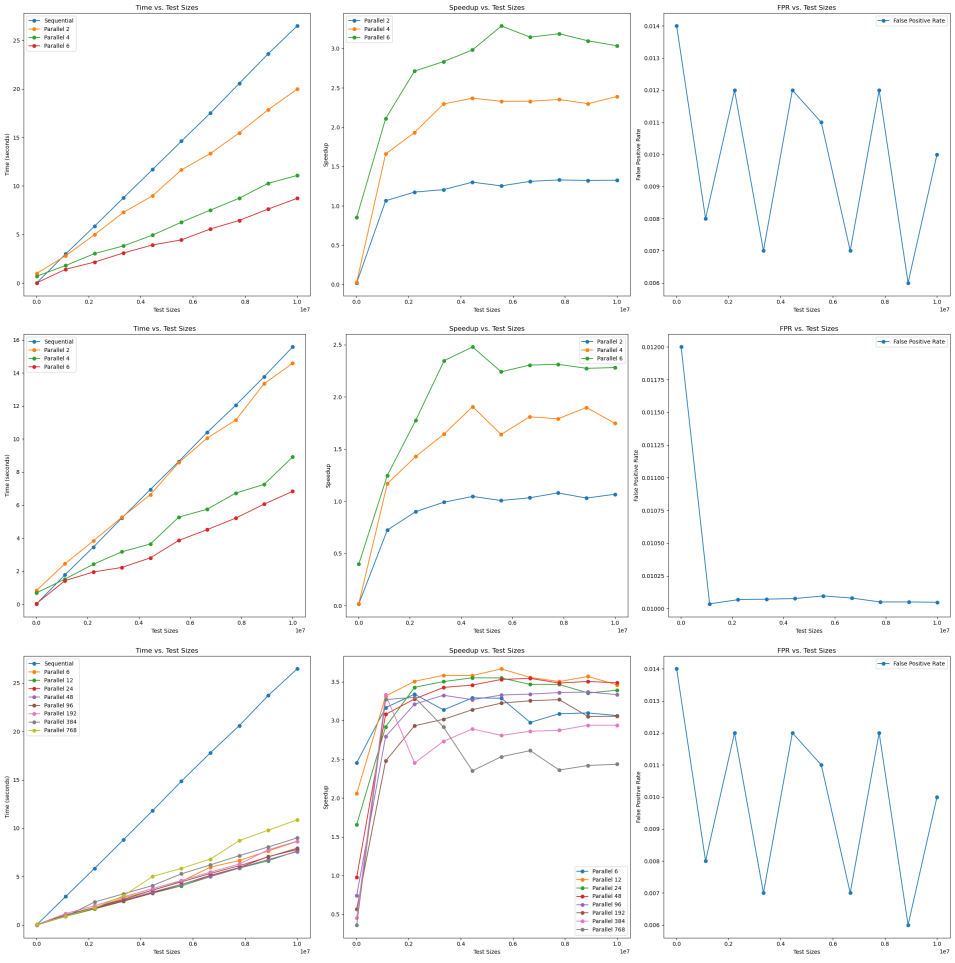

Figure 1: Risultati dei test Joblib

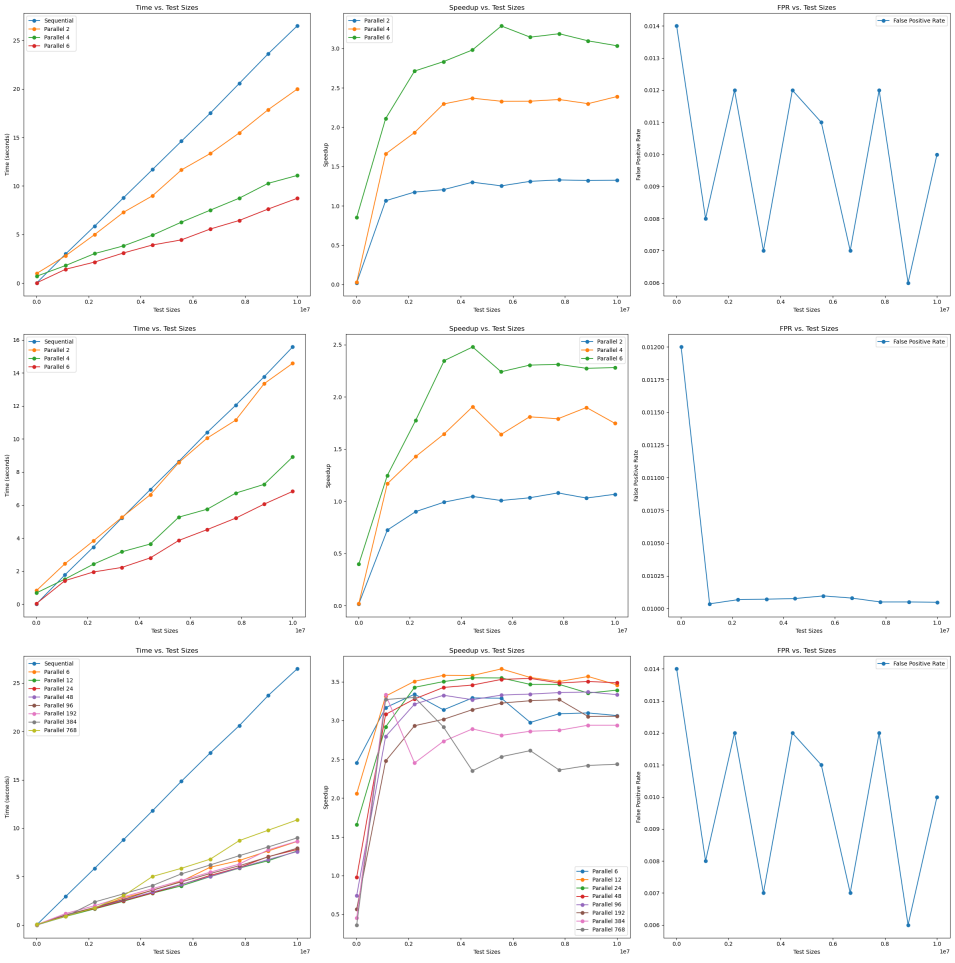

Figure 2: Risultati dei test OMP

5 Conclusioni

Dai risultati ottenuti possiamo vedere come il valore di Speedup massimo si attesti intorno a 3 rispetto alla versione sequenziale. In generale possiamo notare come lo speedup aumenti con l'aumentare della dimensione dell'insieme, fino ad un certo punto per la versione Joblib.

6 Risultati

test	time$_seq$	fpr	time$_par$2	speedup2	time$_par$4
10000	0.020938634872436523	0.014	0.9926893711090088	0.021092836774352062	0.708107
1120000	3.0007448196411133	0.008	2.8176774978637695	1.0649709989578782	1.807430
2230000	5.864649295806885	0.012	4.9905431270599365	1.1751525127610525	3.035002
3340000	8.78724980354309	0.007	7.288600921630859	1.205615439509741	3.829039
4450000	11.690369844436646	0.012	8.9870126247406	1.3008071016005804	4.934762
5560000	14.610514879226685	0.011	11.649624824523926	1.2541618377674886	6.271564
6670000	17.501760721206665	0.007	13.342435121536255	1.3117366179249221	7.510481
7780000	20.55774760246277	0.012	15.463697910308838	1.3294198917813826	8.740763
8890000	23.610797882080078	0.006	17.85670518875122	1.3222370886737622	10.26957
10000000	26.476428508758545	0.01	19.967345714569092	1.3259863823282294	11.07794

test	time$_seq$	fpr	time$_par$2	speedup2
10000	0.015086889266967773	0.012	0.8229348659515381	0.0183330296128
1120000	1.773024082183838	0.010035714285714285	2.4459986686706543	0.7248671493134
2230000	3.4564878940582275	0.010067713004484304	3.8307454586029053	0.9023016359115
3340000	5.21593713760376	0.010071556886227545	5.255182504653931	0.9925320639168
4450000	6.940253496170044	0.0100761797752809	6.627331972122192	1.0472168174710
5560000	8.641323804855347	0.01009658273381295	8.571616649627686	1.0081323230000
6670000	10.4020516872406	0.01007976011994003	10.056608438491821	1.0343498755929
7780000	12.040842771530151	0.01005012853470437	11.13905382156372	1.0809574102443
8890000	13.771966934204102	0.010050281214848144	13.34965181350708	1.0316349165204
10000000	15.570926666259766	0.0100479	14.578480005264282	1.0680761410405

test	time$_seq$	fpr	time$_par$6	speedup6	time$_par$12
10000	0.02573251724243164	0.014	0.010477781295776367	2.455912804059435	0.0125083
1120000	2.9518439769744873	0.008	0.9322412014007568	3.166395105192881	0.8904294
2230000	5.853339195251465	0.012	1.752678394317627	3.339653877304943	1.6692402
3340000	8.812073945999146	0.007	2.8083834648132324	3.137774472897767	2.4592130
4450000	11.797264099121094	0.012	3.583864212036133	3.2917720653312945	3.2947635
5560000	14.849267959594727	0.011	4.515717267990112	3.2883520110646662	4.0501208
6670000	17.783007860183716	0.007	5.976200103759766	2.9756379557966968	5.0010714
7780000	20.590540409088135	0.012	6.667603492736816	3.088147102856058	5.8776726
8890000	23.69925618171692	0.006	7.6488683223724365	3.098400336216817	6.6414175
10000000	26.45124316215515	0.01	8.632525205612183	3.064137379518875	7.6453855

Table 1: Risultati dei test OMP

test	time$_seq$	fpr	time$_par$2	speedup2	time$_par$4
10000	0.020938634872436523	0.014	0.9926893711090088	0.021092836774352062	0.708107
1120000	3.0007448196411133	0.008	2.8176774978637695	1.0649709989578782	1.807430
2230000	5.864649295806885	0.012	4.9905431270599365	1.1751525127610525	3.035002
3340000	8.78724980354309	0.007	7.288600921630859	1.205615439509741	3.829039
4450000	11.690369844436646	0.012	8.9870126247406	1.3008071016005804	4.934762
5560000	14.610514879226685	0.011	11.649624824523926	1.2541618377674886	6.271564
6670000	17.501760721206665	0.007	13.342435121536255	1.3117366179249221	7.510481
7780000	20.55774760246277	0.012	15.463697910308838	1.3294198917813826	8.740763
8890000	23.610797882080078	0.006	17.85670518875122	1.3222370886737622	10.26957
10000000	26.476428508758545	0.01	19.967345714569092	1.3259863823282294	11.07794

test	time$_seq$	fpr	time$_par$2	speedup2
10000	0.015086889266967773	0.012	0.8229348659515381	0.0183330296128
1120000	1.773024082183838	0.010035714285714285	2.4459986686706543	0.7248671493134
2230000	3.4564878940582275	0.010067713004484304	3.8307454586029053	0.9023016359115
3340000	5.21593713760376	0.010071556886227545	5.255182504653931	0.9925320639168
4450000	6.940253496170044	0.0100761797752809	6.627331972122192	1.0472168174710
5560000	8.641323804855347	0.01009658273381295	8.571616649627686	1.0081323230000
6670000	10.4020516872406	0.01007976011994003	10.056608438491821	1.0343498755929
7780000	12.040842771530151	0.01005012853470437	11.13905382156372	1.0809574102443
8890000	13.771966934204102	0.010050281214848144	13.34965181350708	1.0316349165204
10000000	15.570926666259766	0.0100479	14.578480005264282	1.0680761410405

test	time$_seq$	fpr	time$_par$6	speedup6	time$_par$12
10000	0.02573251724243164	0.014	0.010477781295776367	2.455912804059435	0.0125083
1120000	2.9518439769744873	0.008	0.9322412014007568	3.166395105192881	0.8904294
2230000	5.853339195251465	0.012	1.752678394317627	3.339653877304943	1.6692402
3340000	8.812073945999146	0.007	2.8083834648132324	3.137774472897767	2.4592130
4450000	11.797264099121094	0.012	3.583864212036133	3.2917720653312945	3.2947635
5560000	14.849267959594727	0.011	4.515717267990112	3.2883520110646662	4.0501208
6670000	17.783007860183716	0.007	5.976200103759766	2.9756379557966968	5.0010714
7780000	20.590540409088135	0.012	6.667603492736816	3.088147102856058	5.8776726
8890000	23.69925618171692	0.006	7.6488683223724365	3.098400336216817	6.6414175
10000000	26.45124316215515	0.01	8.632525205612183	3.064137379518875	7.6453855

Table 2: Risultati dei test Joblib