센서 네트워크에서 위치 측정을 위한 분산 지지 벡터 머신

문상국*

*목원대학교

Distributed Support Vector Machines for Localization on a Sensor Newtork

Sangook Moon*

*Mokwon University

E-mail : smoon@mokwon.ac.kr

요 약

최근 기계학습 방법을 도입하여 센서 노드에 대한 위치를 파악하는 방법이 관심을 받고 있다. 많은 기계학습 알고리즘 중, 지지벡터머신은 프로그래밍 언어로 구현하기 간편하고, 병렬로 수행이 가능하다. 본 논문에서는 파이썬 프로그래밍 언어로 지지벡터머신을 구현하고, 5대의 라즈베리파이를 사용하여 실험적인 하둡 센서 네트워크와 5개의 노드를 가진 맵리듀스 하둡 소프트웨어 프레임워크를 구성하였다. 기존 지지벡터머신 알고리즘을 분산 처리가 가능하도록 변형하여 위치 측정을 수행하였고, 다양한 파라미터를 변경해가면서 센서 네트워크를 구성하여 효율성, 자원분배, 처리속도를 비교하였다.

ABSTRACT

Localization of a sensor network node using machine learning has been recently studied. It is easy for Support vector machines algorithm to implement in high level language enabling parallelism. In this paper, we realized Support vector machine using python language and built a sensor network cluster with 5 Pi's. We also established a Hadoop software framework to employ MapReduce mechanism. We modified the existing Support vector machine algorithm to fit into the distributed hadoop architecture system for localization of a sensor node. In our experiment, we implemented the test sensor network with a variety of parameters and examined based on proficiency, resource evaluation, and processing time.

키워드

센서네트워크, 지지벡터머신, 위치측정, 하둡

Ⅰ. 서 론

무선통신, 저전력 회로, 반도체 미세 공정의 발달로 인해 차세대 무선 센서 네트워크에 대한 연구는 하나의 독립적인 연구분야로 자리잡고 있다. 무선 센서 노드는 군사 분계 지역의 탐색, 산업 공장의 기기간 통신 등 다양한 분야에 적용하여 임무 수행이 가능하다 [1].

센서 노드의 위치를 파악하는 새로운 방식은 기계학습 [2]에 근거한다. 센서 노드의 위치를 파악하는 데 적용이 가능한 두 종류의 알려진 기계학습 알고리즘은 분류에 의한 방법과 회귀에 의한 방법이다. 분류에 의한 방법에는 k-nearest neighbors, naive Bayes, 지지벡터머신, 결정 트리 등이 있고, 회귀에 의한 방법은 선형 회귀, 지역별 차등적 선형 회귀, ridge, lasso 등이 있다 [3]. Nguyen은 센서 노드의 위치파악이 분류에 의한 방법으로 모델될 수 있다고 밝혔다 [4]. 분류에 의한 방법 중, 지지벡터머신은 낮은 일반화의 에러율, 상대적으로 낮은 계산처리비용, 이분법에 의한 데이터 분석의 단순화라는 특성을 가진다.

우리는 클러스터를 제작하여 센서 노드 위치 측정에 적용 가능한 지지벡터머신 알고리즘을 분산 프로그래밍으로 구현하고 성능을 분석한다.

II. 지지 벡터 머신

기계 학습을 이용한 노드 위치 측정 문제는 결정 가능한 범위 내에서 데이터 포인트가 기준점으로부터 얼마나 떨어져 있는지를 판단하여 이분법적인 결정을 내릴 수 있는 신뢰도가 높은 분류기를 필요로 한다. 그림 1에서 동일한 분포를 보이는 데이터 표본에 대하여 데이터를 서로 다른 두 그룹으로 나눌 수 있는 여러 가지 방법을 보인다. 데이터가 2차원으로 흩어져 있어, 이들을 두 그룹으로 나누기 위해서는 최적으로 분할해주는 1차원 직선이 필요하다. 그림에서는 (D)가 최적이라고 보여지며, 이러한 벡터를 지지 벡터라고 한다.

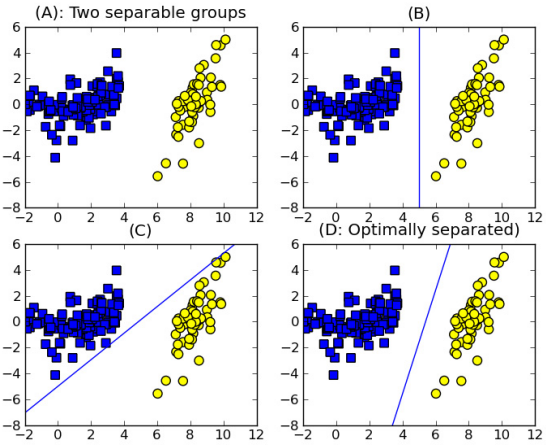

그림 1. 2차원 데이터를 두 그룹으로 나누는 방법

III. 지지벡터를 위한 Pegasos 알고리즘

본 논문에서는, 모든 센서 노드에 대한 위치 측정을 위해 분산 컴퓨팅의 개념을 도입하였다. 기존에 도입되던 순차 최소 최적화 (SMO; sequential minimal optimazation)는 기본적으로 순차적인 방식이다. 만일 SMO의 큰 작업을 작은 부분으로 분할하는 것이 가능하다면, 이에 맵리듀스 방식을 적용하여 분산 처리가 가능하다. Pegasos 알고리즘은 SMO에 대한 대안으로 사용이 가능하며,지지 벡터를 구하는 데 병렬성을 도입할 수 있다.Pegasos에서는 훈련 데이터에서 임의로 선택한 집합이 그룹에 더해지며, 검사를 진행한다. 만일 잘 분류가 되었다면 무시하고 진행하고, 잘 안되었다면 해당 집합을 새로운 집합에 더한다. 작업이 끝나면 가중벡터 w를 잘 분류되지 않았던 데이터의 정보로 갱신하며 이러한 작업은 반복적으로 수행되어 최종적으로 지지 벡터를 얻게 된다. 본 논문에서는 라즈베리파이 Hadoop 클러스터를 활용하기 위해 Pegasos를 python으로 구현하였다. python은 mrjob이라는 라이브러리를 지원하여 분산 프로그래밍이 가능하다. Pegasos에 대한 의사코드를 그림 2에 보인다.

```
Set the weight vector w to 0
for batch in each group
        choose k data points at random
          for data in points
              if (not classified correctly)
                    change w
        accumulate the adjustment to w
```

그림 2. 분산 데이터 노드에서 병렬 수행이 가능한 Pegasos 알고리즘의 의사 코드

IV. Pegasos의 분산화 및 분석

실험 환경을 구축하기 위해, 우리는 mrjob 구조를 사용한 Pegasos 알고리즘을 python 프로그램으로 구현하였다. 맨 먼저, 병렬로 구현이 가능한 부분과 가능하지 않은 부분을 찾아 Pegasos 알고리즘을 Map 단계와 Reduce 단계로 나누었다.

실제 코드를 작성하여 동작하는 코드를 조사하고 실행되는 프로세스를 살펴본 결과 대부분의 시간이 알고리즘 내에서 동작하는 부분에서 소모된다는 것을 파악하였고, 가중치 벡터 w의 축적은 병렬로 처리되지 못한다는 것을 파악하였다. 그래서 또다른 python 라이브러리 중 하나인 pickle을 사용하고 두 개의 반복횟수와 그룹의 크기를 저장하는 새로운 변수를 사용하였다. 또한 mrjob이 어떻게 어떤 순서로 작업을 처리할 것인지를 결정하는 python method과 입출력을 그림 3과 같이 제안하였다. 그림 3에서 보이듯 이 방법은 mapper, mapper_final, reducer 스테이지로 나누어 list로 만들고 반복적으로 조작한다. 정확한 동작을 위해서, mapper는 reducer의 시간에 맞는 결과값을 알 수 있어야 한다.

```
Mapper:
Inputs : {numberofMapper, listofValue}
Outputs : void

Mapper_final:
Inputs : void
Outputs : {l, listofValue}

Reducer:
Inputs : {numberofMapper, listofValue}
Outputs : {numberofMapper, listofValue}
```

그림 3. mrjob이 처리하기 위한 입출력 정의

첫 번째 method인 mapper()는 분산 프로세싱을 처리하기 위한 것이다. 이는 mapper_final()

method가 요구하는 값들을 순서대로 정렬한다. 입력 타입으로는 가중치 벡터, 반복 회수, 인덱스 값을 받는다. 변수의 상태를 저장하는 것이 불가능하기 때문에, 빠르고 쉬운 전달을 위해서 키/값의 형태로 디스크에 저장하는 방식을 택하였다.

두 번째 method인 mapper_final()은 매번 입력 값들이 결정될때마다 동작한다. 이 때 가중치 벡터 w과 인덱스 값 정수 타입 x를 얻게 된다. mapper_final()이 시작하면, 데이터는 라벨과 원시 데이터로 그룹핑 된다. 만일 잘못 분류된 데이터가 존재하는 경우, 그 데이터는 reducer()로 전송된다. 이 때, w와 t의 값이 같이 전달되어 mapper()와 reducer()간의 상태를 조절한다.

마지막 method인 reducer()는 모든 키/값을 반복적으로 판별하여 지역 변수에 전달한다. 프로그램에서 가중치 벡터 w를 갱신하고 지지 벡터 알고리즘의 eta를 결정하기 위해 몇 가지 변수를 더 추가하였다. 새로운 임의의 벡터 그룹이 선택되고 전달되면 값의 키들은 mapper()의 결과가 된다.

라즈베리파이 5기를 사용하여 하둡 플랫폼을 구축하고, 이를 기준으로 클러스터 노드의 개수를 변화시키면서 마스터와 슬레이브 간의 로드 밸런싱을 측정하였다. 먼저, 두 개의 클러스터 노드만으로 Pegasos의 맵리듀스 버전을 실행하였다. 100회 반복 이후, 그림 4와 같은 초평면 위의 데이터 그램을 얻을 수 있었다. 10000개의 샘플 데이터를 입력하였고 반복 회수를 50에서 100까지 변화시켜도 큰 차이를 보이지 않아 한 개의 데이터 샘플을 제외하고는 모두 안정적인 분류를 수행하였다.

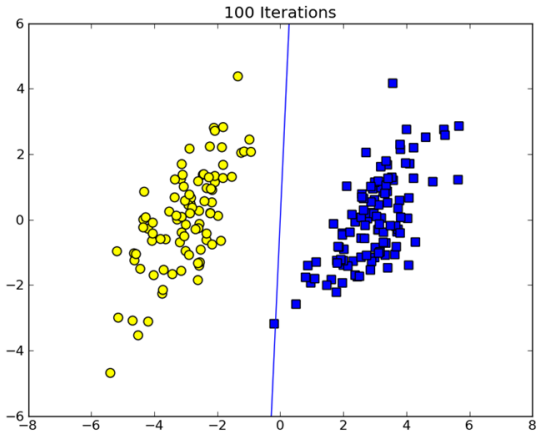

그림 4. 테스트 클러스터에서 python으로 구현한 Pegasos 알고리즘의 실행 결과

V. 결 론

본 논문에서는 초소형 컴퓨터 시스템인 라즈베리파이를 센서 노드로 사용하여 클러스터 플랫폼을 구성하고, Pegasos 알고리즘을 분산 프로그래밍을 지원하도록 변형하여 기계 학습에 의한 센서 노드 위치 측정을 위한 분산 지지 벡터 머신을 구현하였다. 또한, 라즈베리파이를 지지 벡터 머신 용용 센서 노드로 활용하는 것이 적절한지에 대한 분석도 수행하였다. 분산 하둡 클러스터의 센서 노드로 라즈베리파이는 컴퓨팅 파워와 메모리 면에서는 부족한 점을 보여 시간은 비교적 많이 소모하였으나, 정확한 결과를 산출하여 실시간 결과가 필요하지 않은 응용에는 적절히 사용될 수 있을 것으로 기대된다.

감사의 글

이 논문은 2014년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No. 2014R1A1A2A16053925)

참고문헌

[1] L. Zhang, Y. Shao, R. Zhu, J. Yuan, and H. Wang, “Sensor Deployment for Full Detection on Delay Tolerant Event in Hybrid Wireless Sensor Networks,” *Sensor Letters*, Vol. 11, No. 5, pp. 900-909, May 2013.

[2] M. Di and E. M. Joo, “A survey of machine learning in wireless sensor netoworks－from networking and application perspectives,” in *Proceedings of the 6th International Conference on Information, Communications and Signal Processing*, Singapore, pp. 1-5, 2007.

[3] X. Wang, “A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality,” in *Proceedings of the International Joint Conference on Neural Networks,* San Jose, CA, pp. 2351-2358, 2011.

[4] D. A. Tran and T. Nguyen, “Localization in Wireless Sensor Networks based on Support Vector Machines,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, Issue 7, pp. 981-994, July 2008.