**머신러닝 알고리즘 소개**

0.0.0

단 계 계

설계

작성자

홍길동

작성일

2009. .

소 속 속

단 계 계

작성자

엄만진

작성일

2017. 02. 15

**개 정 이 력**

|  |  |  |  |
| --- | --- | --- | --- |
| **버전** | **변경날짜** | **변경내용** | **작성자** |
| 1.0 | 2017.02.15 | 최초 작성 | 엄만진 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**목 차**

[1. 특성추출 5](#_Toc480450597)

[1.1. TF-IDF(Term Frequency-Inverse Document Frequency) 5](#_Toc480450598)

[1.2. 수학적 설명 5](#_Toc480450599)

[1.3. 예제 5](#_Toc480450600)

[1.3.1. 참고자료 8](#_Toc480450601)

[2. 분류(classification) 9](#_Toc480450602)

[2.1. Word2vec 9](#_Toc480450603)

[2.1.1. 개요 9](#_Toc480450604)

[2.1.2. 역사 9](#_Toc480450605)

[2.1.3. 개념 9](#_Toc480450606)

[2.1.4. CBOW(Continous Bag Of Words) 모델 10](#_Toc480450607)

[2.1.5. Skip-gram 12](#_Toc480450608)

[2.1.6. Complexity Reduction 13](#_Toc480450609)

[2.1.6.1. Hierarchical Softmax 13](#_Toc480450610)

[2.1.6.2. Negative Sampling 14](#_Toc480450611)

[2.1.6.3. Further Method : Subsampling Frequent Words 15](#_Toc480450612)

[2.1.7. Spark sample 16](#_Toc480450613)

[2.1.8. 참고자료 16](#_Toc480450614)

[3. 군집화(Clustering) 17](#_Toc480450615)

[3.1. K mean 알고리즘 17](#_Toc480450616)

[3.1.1. 군집화(Clustering) 순서 17](#_Toc480450617)

[3.1.2. 참고자료 18](#_Toc480450618)

1. 특성추출
   1. TF-IDF(Term Frequency-Inverse Document Frequency)

정보 검색과 텍스트 마이닝에서 이용하는 가중치로, 여러 문서로 이루어진 문서군이 있을 때 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내는 통계적 수치이다. 문서의 핵심어를 추출하거나, 검색 엔진에서 검색 결과의 순위를 결정하거나, 문서들 사이의 비슷한 정도를 구하는 등의 용도로 사용할 수 있다.

TF(단어 빈도, term frequency)는 특정한 단어가 문서 내에 얼마나 자주 등장하는지를 나타내는 값으로, 이 값이 높을수록 문서에서 중요하다고 생각할 수 있다. 하지만 단어 자체가 문서군 내에서 자주 사용되는 경우, 이것은 그 단어가 흔하게 등장한다는 것을 의미한다. 이것을 DF(문서 빈도, document frequency)라고 하며, 이 값의 역수를 IDF(역문서 빈도, inverse document frequency)라고 한다. TF-IDF는 TF와 IDF를 곱한 값이다. TF-IDF를 가중치로 사용한다.

요약하면 다른 문서에 나오지 않으면 한 문서에 자주 등장하는 단어가 특징을 나타내는 단어다.

* 1. 수학적 설명

수학적 설명이 필요하면 참고자료의 wikipedia를 보면 된다.

* 1. 예제

다음 예제는 아래 참고 자료에 있는 내용임.

예를 들어 아래와 같은 문서들이 있을 때

|  |  |
| --- | --- |
| Document1 | 은상 정말 멋져 |
| Document2 | 은상 정말 이상해 |
| Document3 | 금상 진짜 멋져 |
| Document4 | 동상 색깔 이상해 |

위 예제에서 단어별 빈도수를 vector 모델로 나타내면 아래 표와 같다. 단순히 전체 단어에 문서마다 나오는 단어수를 표시한것임.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 금상 | 은상 | 동상 | 정말 | 멋져 | 진짜 | 이상해 | 색깔 |
| Document1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| Document2 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| Document3 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Document4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

총 단어들의 빈도수로 IDF를 구하는데 빈도수가 0일 경우 오류가 발생할수 있으므로 모든 빈도수에 1을 더해서 구하기도 함.

|  |  |  |
| --- | --- | --- |
| 단어 | 빈도수 | IDF\* |
| 금상 | 1 | Log2(4/1) = 2 |
| 은상 | 2 | Log2(4/2) = 1 |
| 동상 | 1 | Log2(4/1) = 2 |
| 정말 | 2 | Log2(4/2) = 1 |
| 멋져 | 2 | Log2(4/2) = 1 |
| 진짜 | 1 | Log2(4/1) = 2 |
| 이상해 | 2 | Log2(4/2) = 1 |
| 색깔 | 1 | Log2(4/1) = 2 |

\* IDF는 Log2(전체 문서 수 / 단어가 포함된 문서수)

지금까지 TF와 IDF 값을 구했으니 IF-IDF값을 구할 차례임.(TF x IDF)

각 단어의 TF값에서 동일 단어의 IDF을 곱한 값임.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 금상 | 은상 | 동상 | 정말 | 멋져 | 진짜 | 이상해 | 색깔 |
| Document1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| Document2 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| Document3 | 2 | 0 | 0 | 0 | 1 | 2 | 0 | 0 |
| Document4 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 2 |

유사도를 구하기 위해서는 각 문서의 길이를 구해야 함.

길이는 각 단어의 TF-IDF값을 제곱하여 더한 후 제곱근한 값임.

예를 들어 Document1의 길이는 은상, 정말, 멋져 총 3 단어가 각각 1의 TF-IDF값을 갖고 있으므로 = 1.7 임.

전체 Document의 길이를 구하면 아래와 같다

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 금상 | 은상 | 동상 | 정말 | 멋져 | 진짜 | 이상해 | 색깔 | 길이 |
| Document1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1.7 |
| Document2 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1.7 |
| Document3 | 2 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 3 |
| Document4 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 2 | 3 |

다음은 입력한 쿼리가 4개의 Document중 가장 유사한 Document를 찾는 방법이다.

입력한 쿼리가 “은상 진짜 은상” 인 경우 쿼리의 vector모델을 구하면 아래와 같다.

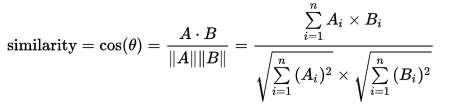
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 금상 | 은상 | 동상 | 정말 | 멋져 | 진짜 | 이상해 | 색깔 | 길이 |
| 쿼리 | 0 | 1 | 0 | 0 | 0 | 2 | 0 | 0 | 2.23 |

TF-IDF 값 : (전체 문서중 은상이 포함된 문서 수 / 쿼리의 은상 수) \* 1(은상 IDF 값)

길이 : = 2.23

이렇게 구해진 vector 모델을 기반으로 쿼리문과 각 Document들 간의 검색어의 거리를 측정하면 된다.

보통 거리 측정에는 유클리드 거리나 코사인 거리를 이용하는 경우가 많음.아래는 코사인거리를 이용한 결과임.



위 수식에 대입하면 되는데 A를 쿼리라고 하고 B를 각각의 Document라고 했을 때 계산된 값은 아래와 같다.

cos(q, document 1) = (1\*1)/(2.23\*1.7) = 0.263

cos(q, document 2) = (1\*1)/(2.23\*1.7) = 0.263

cos(q, document 3) = (2\*2)/(2.23\* 3) = 0.597

cos(q, document 4) = (0)/2.23\*3) = 0

**코사인인 유사도(Cosine Similarity)**

<http://euriion.com/?p=548>

**유크리드 거리**

http://dyndy.tistory.com/159

* + 1. 참고자료

<https://ko.wikipedia.org/wiki/TF-IDF>

<http://meoyameoya.tistory.com/3>, <http://meoyameoya.tistory.com/4>

1. 분류(classification) 와 회귀(Regression)
   1. Word2vec
      1. 개요

Word2Vec는 사람이 알고 있는 단어를 컴퓨터가 알수 있게 수치화(vector로 변환)하는 방법이다.

Vector로 변환 이점

* 수치화 되어서 연산 가능
* 단어들 사이 유사도 측정 가능.
* vector연산을 통한 추론 가능.

예전에는 NLP(Neural Language Processing)에서는 ‘one-hot encoding’ 방식을 많이 사용했다.

이런 방식은 email 내용의 단어를 수치화 하여 스팸 여부를 판단하는 스팸 분류기에 적용되었었다.

**One-hot encoding**

One-hot encoding은 단어별로 vector 값을 설정하는 방식이다.

예를 들어 [감자, 딸기, 수박, 사과] 라는 단어를 갖고 있는 사전이 있다고 가정할 때 4짜리 vector를 만들고 그 단어가 해당 되는 자리에 1을 넣고 나머지는 0을 넣는 방식이다.

수박을 표시하는 vector는 [0, 0, 1, 0] 이다.

One-hot encoding 방식의 문제점은 다른 단어와의 차이점을 알수가 없다는 점이다.

* + 1. 역사

단어를 vector화 하는 노력은 1980년대부터 연구가 꾸준히 진행 되었다. 2000년대에 획기적인 NNLM(Neural Network Language Model : 신경망 언어 모델) 방법이 나왔으며 속도가 느리다는 문제가 있어 RNNLM(Recurrent Neural Network Language Model : 회귀 신경망 언어 모델) 방법으로 발전 되었다. 2013년에 CBOW(Continuos Bag Of Words)와 Skip-gram이라는 아키텍로 발전되었으며 오느날 word2vec의 모양새로 이어 졌다.

* + 1. 개념

단어의 vector화에는 가장 중요한 가정이 있다. 그 가정은 ‘모든 Word Embedding관련 학습들은 기본적으로 언어학의 Distributional Hypothesis 라는 가정에 입각하여 이루어 진다’ 이다. 쉽게 말해 비슷한 분포를 가진 단어들(같은 문맥에 등장하는 단어들)은 비슷한 의미를 가진다는 의미이다.

2013년 구글에서 발표된 연구로 continuous Word Embedding 학습 모형이다.

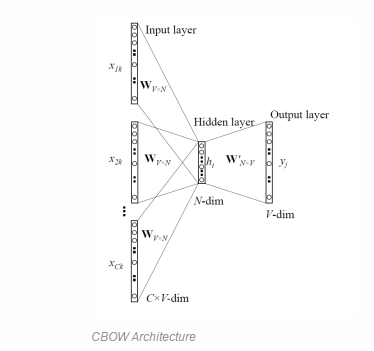
학습을 시키기 위해 두가지의 네트워크 모델을 제시 하였다. 그 모델이 CBOW, Skip-gram 모델이다.

**Word Embedding**

고차원의 데이터를 그보다 낮은 차원으로 변환하면서 모든 데이터간의 관계가 성립되도록 처리하는 과정을 말한다.  
컴퓨터는 숫자만 인식할 수 있고, 한글이나 그림 등의 컴퓨터가 다루는 모든 것들은 반드시 숫자로 변환되어야 한다.  
Word Embeddins To Vector를 직역하면 "단어를 벡터(숫자)로 삽입하기" 정도가 된다. 다시 말해, 문자로 이루어진 단어를 숫자로 변환하는 것을 word embedding이라고 부른다.  
단어 자체를 아스키코드나 유니코드로 처리해서 지금까지 잘 사용해 왔지만, 이것만 갖고는 의미를 추론할 수가 없다.  
왕과 여왕이 관련 있는 단어이고, 왕은 남자, 여왕은 여자라는 사실을 인지하지는 못한다. 왕에서 남자를 빼면 여왕이 될 수도 있는데, 이걸 하고 싶은 것이다. 수없이 많은 단어들 중에서 '파이썬'이라는 단어와 가까운 단어가 무엇인지 알고 싶기 때문에 단어를 숫자로 변환할 필요가 있었다.  
word embedding에는 초기 모델인 NNLM과 RNNLM이 있고, 최근에 발표됐고 현재 가장 많이 사용하는 Word2Vec 모델은 CBOW와 Skip-Gram 모델을 사용해서 학습되어진다.  
word embedding 프로세스에는 신경망, 차원감소, 확률적 모델, 문맥상 표현 등의 여러 가지 처리 과정들이 포함된다.

* + 1. CBOW(Continous Bag Of Words) 모델

“집 앞 편의점에서 아이스크림을 사 먹었는데, \_\_\_ 시려서 너무 먹기가 힘들었다.” 라는 문장에서, 사람들은 \_\_\_ 부분에 들어갈 단어가 정확하게 주어지지 않아도 앞 뒤의 단어들을 통해 ‘이가’ 라는 말이 들어갈 것을 추측할 수 있다. CBOW 모델도 마찬가지의 방법을 사용한다. 주어진 단어에 대해 앞 뒤로 C/2개 씩 총 C개의 단어를 Input으로 사용하여, 주어진 단어를 맞추기 위한 네트워크를 만든다.



CBOW 모델은 크게 Input Layer, Projection Layer, Output Layer로 이루어져 있다. 그림에는 중간 레이어가 Hidden Layer라고 표시되어 있기는 하지만, Input에서 중간 레이어로 가는 과정이 weight를 곱해주는 것이라기 보다는 단순히 Projection하는 과정에 가까우므로 Projection Layer라는 이름이 더 적절할 것 같다. Input Layer에서 Projection Layer로 갈 때는 모든 단어들이 공통적으로 사용하는 VxN 크기의 Projection Matrix W가 있고 (N은 Projection Layer의 길이 = 사용할 벡터의 길이), Projection Layer에서 Output Layer로 갈 때는 NxV 크기의 Weight Matrix W’ 가 있다.  (주의해야할 점은, 두 행렬은 별개의 행렬이라는 점이다. 예전 What’s Cooking 대회를 할 때는 이 W’ 가 W의 transpose라고 생각하고 구현했었다. 당시에 어떻게든 봐줄만한 결과가 나왔던 것이 신기하다..) Input에서는 NNLM 모델과 똑같이 단어를 one-hot encoding으로 넣어주고, 여러 개의 단어를 각각 projection 시킨 후 그 벡터들의 평균을 구해서 Projection Layer에 보낸다. 그 뒤는 여기에 Weight Matrix를 곱해서 Output Layer로 보내고 softmax 계산을 한 후, 이 결과를 진짜 단어의 one-hot encoding과 비교하여 에러를 계산한다.

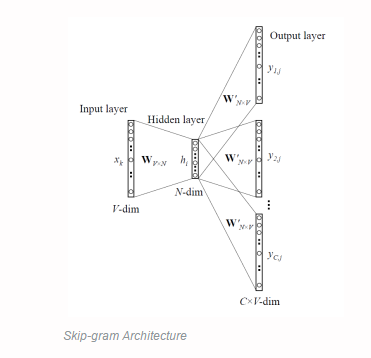
CBOW 모델에서 하나의 단어를 처리하는 데에 드는 계산량은 다음과 같다.

* C개의 단어를 Projection 하는 데에 C x N
* Projection Layer에서 Output Layer로 가는 데에 N x V

따라서 전체 계산량은 CxN + NxV가 되는데, 앞서 말한 V를 ln V로 줄이는 테크닉을 사용하면 전체 계산량이 CxN + N x lnV가 된다. 이 식을 잘 보면, 이 모델이 앞서 다른 모델들에 비해 왜 계산이 빠른지 알 수 있다. CBOW 모델에서 보통 C는 10 내외의 크기로 잡으므로, 전체 계산량은 결국 Projection Layer의 크기 N과 log-사전크기의 lnV의 크기의 곱에 비례하게 된다. 즉, C=10, N=500, V=1,000,000으로 잡아도 500 x (10+ln(1,000,000)) = 약 10000의 계산량밖에 들지 않는 것이다. 이는 앞서 확인한 NNLM이나 RNNLM에 비해 정말 엄청나게 줄어든 계산량이라는 것을 확인할 수 있다.

* + 1. Skip-gram

Skip-gram 모델은 CBOW와는 반대 방향의 모델이라고 생각할 수 있을 것 같다. 현재 주어진 단어 하나를 가지고 주위에 등장하는 나머지 몇 가지의 단어들의 등장 여부를 유추하는 것이다. 이 때 예측하는 단어들의 경우 현재 단어 주위에서 샘플링하는데, ‘가까이 위치해있는 단어일 수록 현재 단어와 관련이 더 많은 단어일 것이다’ 라는 생각을 적용하기 위해 멀리 떨어져있는 단어일수록 낮은 확률로 택하는 방법을 사용한다.



Skip-gram 모델에서 하나의 단어를 처리하는 데에 드는 계산량은 다음과 같다. C개의 단어를 샘플링했다고 할 때,

* 현재 단어를 Projection 하는 데에 N
* Output을 계산하는 데에 N x V, 테크닉을 사용하면 N x ln V
* 총 C개의 단어에 대해 진행해야 하므로 총 C배

로 총 C(N + N x lnV) 만큼의 연산이 필요하다. 즉 이 모델도 CBOW 모델같이 N x lnV에 비례하는 계산량을 가진 모델이기는 하지만, 몇 개의 단어를 샘플링하냐에 따라서 계산량이 그것에 비례하여 올라가게 되므로 CBOW 모델에 비해서는 학습이 느릴것이다.

CBOW 모델과 Skip-gram 모델을 비교하면 CBOW의 모델이 조금 더 논리적이라고 개인적으로 생각하지만, 실제 실험 결과에서는 Skip-gram이 CBOW에 비해 전체적으로 다소 좋은 결과를 내는 추세를 보인다. 현재는 대부분의 사람들이 Skip-gram 모델을 사용하는 것 같다.

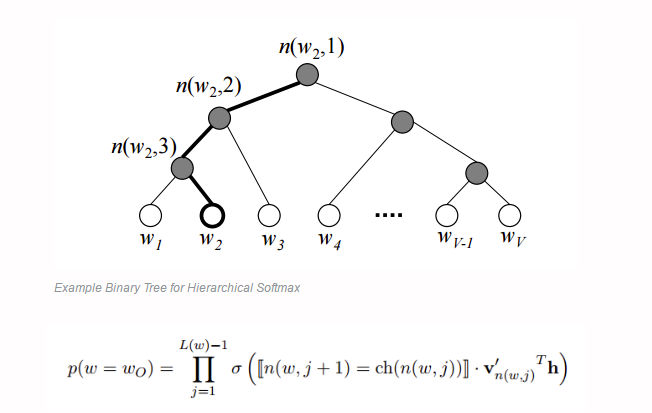
* + 1. Complexity Reduction

CBOW와 Skip-gram 모델에서 V의 데이터 수에 따라 처리 속도가 느려진다. 영어 단어는 백만개가 넘는데 네트워크의 Output Layer에서 softmax 계산을 하기 위해서는 각 단어에 대해 전부 계산을 해서 normalization을 해줘야 하고 이에 따라 추가 연산이 늘어나므로 연산이 느려진다.

이를 해결하기 위해 아래 방법을 사용한다.

* + - 1. Hierarchical Softmax

Hierarchical Softmax는 계산량이 많은 Softmax function 대신 보다 빠르게 계산가능한 multinomial distribution function을 사용하는 테크닉이다. 이 방법에서는 각 단어들을 leaves로 가지는 binary tree를 하나 만들어놓은 다음(complete 할 필요는 없지만, full 할 필요는 있을 것 같다), 해당하는 단어의 확률을 계산할 때 root에서부터 해당 leaf로 가는 path에 따라서 확률을 곱해나가는 식으로 해당 단어가 나올 최종적인 확률을 계산한다.



* L(w)는 w라는 leaf에 도달하기 까지의 path의 길이를 의미한다.
* n(w, i) 는 root에서부터 w라는 leaf에 도달하는 path에서 만나는 i번째 노드를 의미한다. n(w, 1)은 루트가 될 것이고, n(w, L(w)) 는 w가 될 것이다.
* ch(node) 는 node의 고정된 임의의 한 자식을 의미하며, 여기서는 단순하게 node의 왼쪽 자식이라고 생각하겠다. 이렇게 생각해도 무방하다.
* [[x]] 는 x가 true일 경우 1, false일 경우 -1을 반환하는 함수로 정의한다.
* Hierarchial Softmax를 사용할 경우 기존 CBOW나 Skip-gram에 있던 W’ matrix를 사용하지 않게 된다. 대신, V-1개의 internal node가 각각 길이 N짜리 weight vector를 가지게 된다. 이를 v’\_i 라고 하고 학습에서 update 해준다.
* h는 hidden layer의 값 벡터이다.
* sigma(x)는 sigmoid function 1/(1+exp(-x)) 이다.

이렇게 정의하고 위의 식과 같은 방법으로 p(w)를 계산한다고 해보자. 이 경우 각 스텝마다 길이 N짜리 벡터 두 개의 내적이 일어나므로 계산량 N이 필요하며, binary tree를 잘 만들었을 경우 평균적으로 루트로부터 leaf까지의 거리는 평균 O(lnV)일 것이므로 총 N x lnV의 계산량만으로 특정 단어가 나올 확률을 계산할 수 있다. 네트워크의 Error Function을 Categorical Cross-entropy 로 사용할 경우 최종 Error를 계산하기 위해서 해당하는 단어에 대해서만 확률을 계산하면 되므로, 다른 추가적인 연산 없이 O(N x lnV)의 계산량만으로 계산이 끝난다.

또한 특정 hidden layer 값에 대해 모든 단어들의 확률을 더한 sigma p(w\_i | hidden layer) 를 생각해보자. Full binary tree를 산정하고 v\_node와 h의 내적 값을 x라고 생각할 경우,

* 특정 node에서 왼쪽 자식으로 갈 때의 확률 = sigmoid(x)
* 특정 node에서 오른쪽 자식으로 갈 때의 확률 = sigmoid(-x) = 1-sigmoid(x)

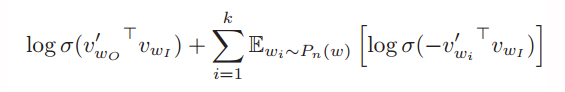
따라서 특정 노드에서 왼쪽, 오른쪽 자식으로 갈 확률을 더하면 1이 된다. 이를 이용하면, sigma p(w\_i | hidden layer) 값이 1이라는 것을 쉽게 보일 수 있을 것이다. Softmax function의 계산이 오래 걸렸던 것은 확률 계산을 위해 모든 결과에 대한 합을 1로 만들어주기 위함이었다. 이 과정에서 최종적으로 나온 output값에 대해 일일히 계산을 해주어서 전체 합으로 normalize를 해주었기 때문에 V 만큼의 계산이 더 필요했던 것인데, 이 Hierarchical Softmax를 사용하면 전체 확률에 대한 계산 없이 전체 합을 1로 만들어 줄 수 있어 좋은 multinomial distribution function으로 사용할 수 있게 되는 것이다.

word2vec 논문에서는 이러한 Binary Tree로 Binary Huffman Tree를 사용했다고 한다. Huffman Tree를 사용할 경우 자주 등장하는 단어들은 보다 짧은 path로 도달할 수 있기 때문에 전체적인 계산량이 더 낮아지는 효과를 볼 수 있을 것이다. 또한 Huffman Tree는 Full Binary Tree이기 때문에 Hierarchical Softmax의 조건에도 부합한다.

* + - 1. Negative Sampling

Hierarchical Softmax의 대체재이며 전체 단어들에 대해 계산하는 방식에서 일부만 추출하여 softmax를 계산하고 normalization하는 방식이다. 이 때 실제 target으로 사용하는 단어의 경우 반드시 계산을 해야하므로 이를 ‘positive sample’ 이라고 부르고, 나머지 ‘negative sample’ 들을 어떻게 뽑느냐가 문제가 된다. 이 뽑는 방법을 어떻게 결정하느냐에 따라 Negative sampling의 성능도 달라지고, 이는 보통 실험적으로 결정한다.

그런데 word2vec의 경우 기존과는 다른 새로운 Error Function을 정의해서 사용한다. 그들은

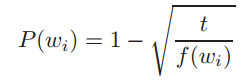


다음과 같은 목표를 maximize 하도록 weight을 조정한다. 이 때 좌측 항은 positive sample에 대한 항이고, 우측은 뽑은 negative sample들에 대한 항이다. 왜 이러한 objective function을 사용하게 되었는지는 논문 [5]에 자세하게 설명되어 있다. 기본적으로 보고있는 단어 w와 목표로 하는 단어 c를 뽑아서 (w,c)로 둔 후, positive sample의 경우 ‘이 (w,c) 조합이 이 corpus에 있을 확률’ 을 정의하고, negative sample의 경우 ‘이 (w,c) 조합이 이 corpus에 없을 확률’을 정의하여 각각을 더하고 log를 취해서 정리하면 위과 같은 형태의 식이 나온다. 자세한 수리적 전개 과정을 보고싶은 분들은 논문 [5]를 참조하시면 되겠다.

보통 Negative Sampling에서 샘플들을 뽑는 것은 ‘Noise Distribution’ 을 정의하고 그 분포를 이용하여 단어들을 일정 갯수 뽑아서 사용하는데, 논문에서는 여러 분포를 실험적으로 사용해본 결과 ‘Unigram Distribution의 3/4 승’ (여기서 Unigram Distribution은 단어가 등장하는 비율에 비례하게 확률을 설정하는 분포라고 할 수 있다. 이 경우 각 확률을 3/4승 해준 후, Normalization factor로 나누어서 단어가 등장할 확률로 사용한 것이다) 을 이용한 분포를 이용해서 실험한 결과 unigram, uniform 등 다른 분포들보다 훨씬 좋은 결과를 얻을 수 있었다고 한다.

* + - 1. Further Method : Subsampling Frequent Words

Hierarchical Softmax와 Negative Sampling은 모델 자체의 계산복잡도를 줄이기 위한 테크닉이었다. 여기서 추가적인 방법으로, 논문에서는 ‘the’, ‘a’, ‘in’ 등 자주 등장하는 단어들을 확률적으로 제외하여 학습 속도를 향상시켰을 뿐만 아니라 성능까지 향상시켰다. 단어 w의 등장 빈도를 f(w)라고 할 때, 학습할 때 각 단어는



의 확률로 제외된다. 이 때 t는 빈도가 일정값 이상일 때만 제외하겠다는 느낌의 threshold 값인데, 논문에서는 10^-5 의 값을 사용했을 때 가장 좋은 결과를 얻을 수 있었다고 한다.

* + 1. Spark sample

<https://github.com/apache/spark/blob/master/examples/src/main/java/org/apache/spark/examples/ml/JavaWord2VecExample.java>

* + 1. 참고자료

<https://shuuki4.wordpress.com/2016/01/27/word2vec-%EA%B4%80%EB%A0%A8-%EC%9D%B4%EB%A1%A0-%EC%A0%95%EB%A6%AC/>

<http://pythonkim.tistory.com/92>

softmax : <http://blog.naver.com/PostView.nhn?blogId=ishkkman&logNo=220967416333>

NNLM(Neural Network Language Model) : <https://brunch.co.kr/@goodvc78/16>

CBOW(Continuous Bag Of Word) : <http://operatingsystems.tistory.com/136>

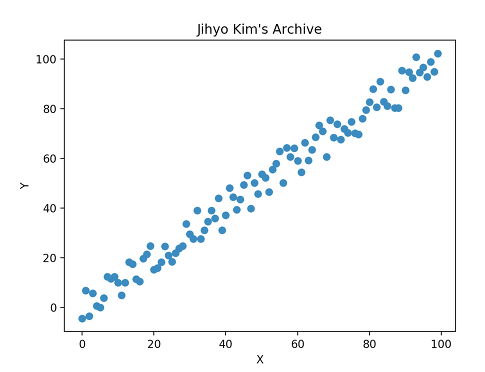
용어 정리 : http://pythonkim.tistory.com/92

batch normalization : <https://shuuki4.wordpress.com/2016/01/13/batch-normalization-%EC%84%A4%EB%AA%85-%EB%B0%8F-%EA%B5%AC%ED%98%84/>

* 1. 선형회귀
     1. 개념

사전적 의미는 위키 백과에 잘 설명되어 있다.

예를 들어 설명하면



X 축을 집의 평수, Y 축을 집 매매 가격이라고 가정할 때 100평이 넘는 가격을 예측할 때 사용되어 진다. (고등 수학을 공부한)우리는 위 그래프 보고 y=x 그래프를 생각할 수 있다. 그래서 평수(x값)를 입력하면 가격(y 값)을 구할수 있다.

그럼 컴퓨터가 y=x라는 그래프라고 인식하기 위해서는 어떻게 해야하는지 그 방법에 대해 알아본다.

* + 1. 절차(손실함수)

1. 임의의 선형함수를 설정한다 y = Wx + B (임의의 W와 B)

2. 임의의 선형함수에 x를 대입한다.

3. 나온 값을 실제 y 값과 비교한다.

4. 두 값의 오차를 구한다.

5. 오차을 최소화 하는 W와 B를 구한다

* + 1. 참고자료

<http://jihyokimarchive.tistory.com/2>

<http://daryan.tistory.com/23>

위키백과 : <https://ko.wikipedia.org/wiki/%EC%84%A0%ED%98%95_%ED%9A%8C%EA%B7%80>

1. 군집화(Clustering)
   1. K mean 알고리즘

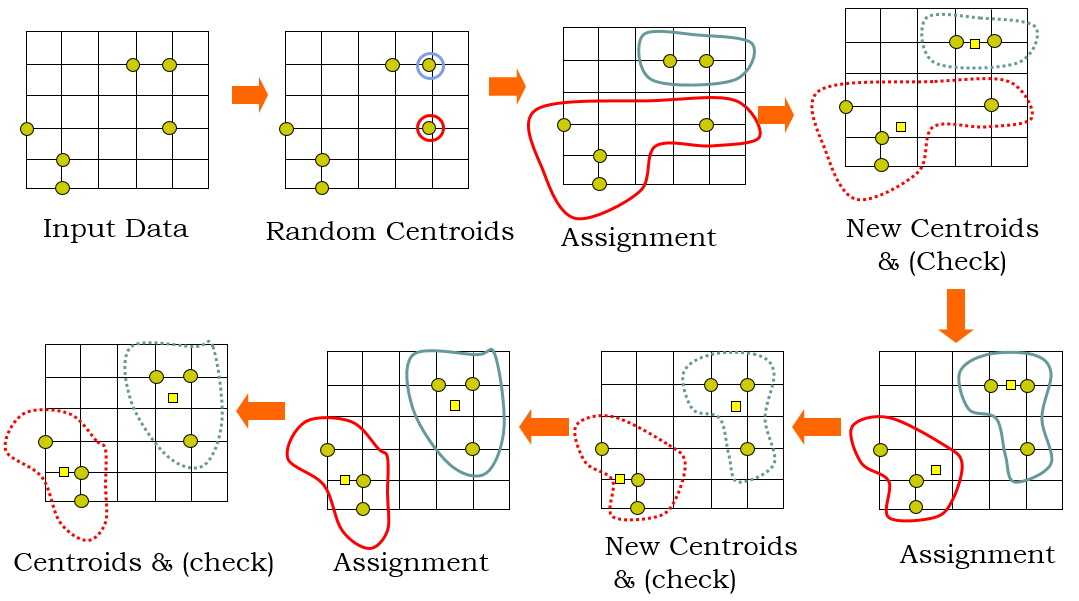
K mean 알고리즘이란 k개의 평균(means) 벡터를 이용한 군집화(clustering) 알고리즘이다.

* + 1. 군집화(Clustering) 순서

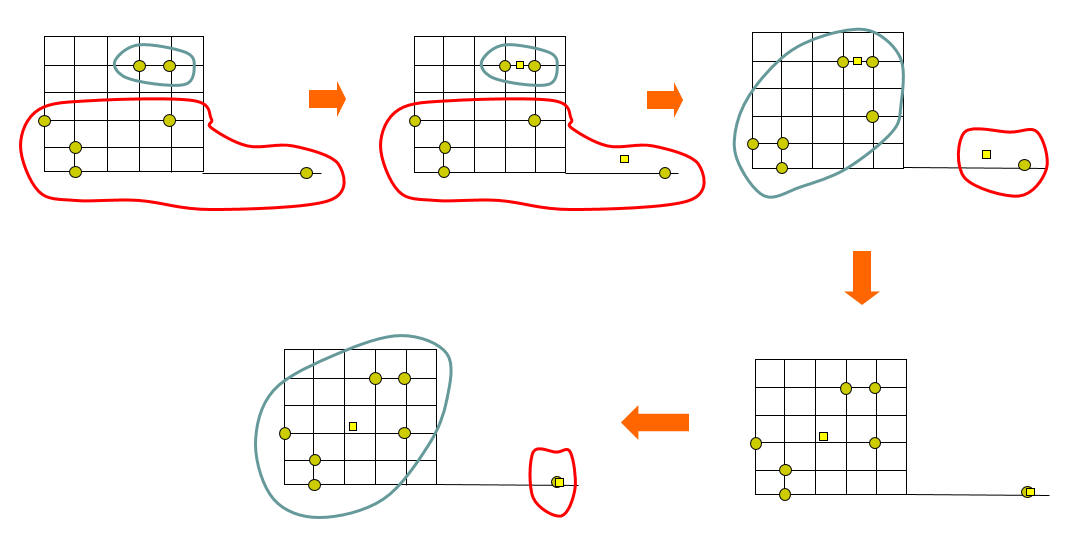
clustering 순서는 아래와 같고 더 이상 진행이 어려울때까지 진행한다.

1. 처음 중심값(centroid) k개 선택
2. 선택된 중심값 k개로부터 데이터간의 거리(distance)를 측정.
3. 가장 가까운 데이터들을 할당(assign)한다.
4. 각 군집(clustering)의 새로운 중심값 선정.
5. 중심값 변화가 있을 경우 2번부터 반복

위 순서를 그림으로 표시한것이다.



위 그림과 같이 두개의 군집으로 구성된 상태에서 새로운 데이터(noise)가 추가되면 아래와 같이 새로운 군집화가 이루어 진다.



* + 1. 참고자료

<http://ai-times.tistory.com/158>

<http://stanford.edu/~cpiech/cs221/handouts/kmeans.html>

<http://jangjy.tistory.com/47>