대용량 온라인 필기 한자 인식을 위한 구조 코드 및 HMM 기반의 클러스터링 방법

김광섭 ○, 하진영

강원대학교 컴퓨터정보통신공학과

kwangseob@kangwon.ac.kr, jyha@kangwon.ac.kr

Clustering Method based on Structure Code and HMM for Huge Class On-line Handwritten Chinese Character Recognition

Kwang-Seob Kim○, Jin-Young Ha

Dept. of Computer Engineering, Kangwon National University

요 약

본 논문에서는 은닉 마르코프 모델(HMM)을 기반한 대용량의 필기 한자 인식의 문제점인 시스템 리소스의 한계와 인식에 소요되는 많은 시간을 단축하기 위해 구조코드와 HMM에 최적화 된 클러스터링 알고리즘을 제안한다. 제안하는 클러스터링 알고리즘의 기본 개념은 훈련된 HMM를 대상으로 하고, HMM의 파라미터 수가 동일한 클래스에 대해서 클러스터를 구성하는 것이다. 또한 인식에 소요되는 시간을 줄이기 위해 2단계 클러스터모델 구조를 사용한다. 총 98,639 종류의 일본 한자를 대상으로 한 실험에서 평균 0.92 sec/char 인식 속도와 30순위 후보인식률 96.03%를 보임으로서 대용량 필기 한자 인식을 위한 좋은 방안이 될 것이라 기대한다.

1. 서 론

HMM(hidden Markov model)은 1980년대 후반부터 음성 인식과 문자 인식 분야에 적용되어 좋은 결과를 내고 있다. 이러한 HMM의 성공은 높은 모델링 능력과 EM-알고리즘과 같은 주어진 모델 구조에 맞춰 모델 파라미터를 재추정할 수 있는 강력한 훈련 알고리즘이 기인한 바가 크다. 이러한 장점으로 인해 HMM은 연구 목적뿐만 아니라 상당수의 상업용 시스템에서 사용되어 좋은 성능을 보이고 있다. 하지만 HMM은 인식 대상 클래스 수가 대용량일 경우에는 인식 속도가 느려지고, 필요한 메모리가 매우 많아지는 문제 등으로 인해, 시스템 최적화를 위한 해법이 필요하다.

이에 본 논문에서는 대용량 온라인 필기 한자 인식을 위해 문자 단위로 훈련된 HMM을 클러스터링하는 방법을 제안한다. 본 논문에서는 약 10만 자에 이르는 한자를 대상으로 삼았고, 2단계 클러스터모델 구조로 만들어 메인 메모리의 사용량을 축소하고, 빠른 시간 내에 응답 결과를 얻을 수 있도록 하였다.

본 논문의 구성은 다음과 같다. 2절에서는 사용자로 하여금 입력된 필기 데이터로부터 특징 추출을 하여 구조 코드를 만드는 방법과 HMM을 훈련하는 방법을 소개하고, 3절에서는 훈련된 HMM을 클러스터링하여 2단계 클러스터모델 구조로 만드는 방법에 대해서 기술한다. 4절에서는 계층화된 HMM 클러스터 구조에서 효율적으로 인식하는 방법과 인식범위를 축소하는 방법에 대해 기술하고, 구현방법에 대해 언급한다. 5절에서는 실험 및 결과 분석을 제시하고 6절에서는 결론을 맺는다.

2. HMM 훈련

2.1 전처리

사용자의 입력을 통해 얻어진 필기 데이터는 사용자의 글씨를 쓰는 습관이나 속도 또는 실수, 그리고 기타 다른 환경적인 요인에 영향을 받은 것이다. 그러므로 보다 높은 인식률을 위해서는 이를 보정할 필요가 있기 때문에 전처리 과정이 필요하다[1]. 본 논문에서는 평활화 (smoothing), 훅 제거 (hook elimination), 난폭점 교정 (wild point correction), 획의 크기 정규화(normalize) 등이 사용되었다.

2.2 구조코드(Structure Code)

구조 코드 열은 과거 온라인 한글인식에서 사용하던 방법으로 특징 점의 좌표를 추출하여 특징 벡터를 생성 후 이를 Clustering 하여 64개의 구조 코드를 생성한다. 특징 벡터는 연속된 자취로 구성된 하나의 획에서 갑작스런 각의 변화가 일어나는 부분과 각 획의 시작과 끝점을 특징 점으로 잡고 특징 점 사이의 부분 획에 대해 생성하였다. 특징 벡터의 요소는 총 다섯 개로 표 1과 같다[2].

표 1. 특징 벡터

종류	내 용
∨1 distance	특징 점 사이의 누적거리
∨2 straightness	부분 획의 곧은 정도
∨3 direction	시작점에서 끝점으로 향하는 각도
∨4 real/imaginary stroke	실제 획과 가상 획 실제 획 = 1, 가상 획 = 0

∨5 Rotation	부분 획의 굴곡 방향 시계방향 = 1, 반시계 = -1 무 방향 = 0

부분 획에서 특징 벡터를 추출한 후 K-Means 클러스터링 알고리즘을 사용하여 총 64개의 클러스터를 생성하였다[3,4].

추출된 벡터들에 대해 K-Means 클러스터링 알고리즘에 적용할 때의 클러스터 센터와 입력 벡터 사이의 거리 계산 식과 벡터의 가중치는 식 (1)과 같다.

$$dist = \sum_{i=1}^{5} W_i * | Input_i - Center_i |$$

$$where \quad Input_i = ith\ element\ of\ Input\ Vector$$

$$Center_i = ith\ element\ of\ Cluster\ Center$$

$$W_i = ith\ Weight$$

$$W_i = \{ 1, 15, 10, 100000, 100 \}$$

식 1.

K-Means 클러스터링을 통해 생성된 64개의 클러스터의 센터값은 새로운 입력벡터를 관측열로 변환시킬 때 사용한다[4,5].

2.3 HMM의 구조

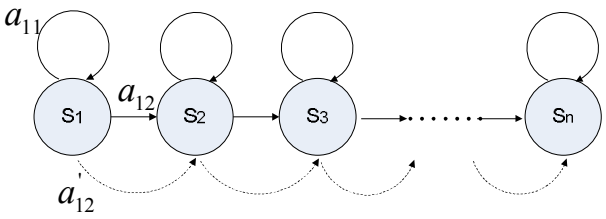

그림 1. Left-to-right HMM

그림 1는 본 논문에서 사용한 left-to-right HMM의 구조를 보여주고 있다. $S_{1\sim n}$은 모델의 상태 수를 N이라 하였을 때 각각의 상태를 나타낸다. a_{ij}는 상태 i에서 상태 j로의 전이하는 non-null transition을 의미하고, a'_{ij}는 관측열 없이 전이 가능한 null-transition을 의미한다.

2.4 HMM의 훈련

HMM의 훈련을 위해 충분히 많은 한자 구조 코드 열 데이터를 확보하고, 수집된 데이터를 사용한 모델의 파라미터 훈련은 Baum-Welch 알고리즘을 사용하였다. 훈련 모델의 상태 수는 모델의 구조 코드 열의 평균 길이에 의존하도록 가변적으로 적용하였고, 너무 작은 구조 코드 열의 길이를 갖는 모델의 경우 상태 수를 '2'로 하한선을 두었고, 반대로 너무 큰 구조 코드 열의 길이를 갖는 모델의 경우 상태 수를 '70'으로 상한선을 두었다.

3. 클러스터링(Clustering)

3.1 단계 1 클러스터링

본 논문에서 제안하는 클러스터링 알고리즘의 골자는 훈련된 HMM이 클러스터링의 대상이 되고, 동일한 파라미터 수를 갖는 HMM 집합에 대해서 개별적으로 클러스터링이 이뤄진다는데 있다. 또한 개별 클러스터 원소 개수의 크기에 비례하도록 분할 (split)과 합병 (merge)을 하여 서로 다른 동일한 상태 수를 갖는 HMM 집합들에 대해서 적합한 클러스터 개수를 개별적으로 구한다. 각각의 집합의 정의는 다음과 같다.

V_{PN} : V is Set of Model, 2 <= PN <= 70, PN : parameter

E_N : E is element of V_{pn}, N is total element number

C_N : C is Cluster of E, N is Cluster's sequence

단계 1 클러스터링은 동일한 상태 수를 갖는 HMM들에 대해서 진행한다는 전제조건을 만족하기 위해 $V_{P2}\sim V_{p70}$에 대해 개별적으로 진행된다.

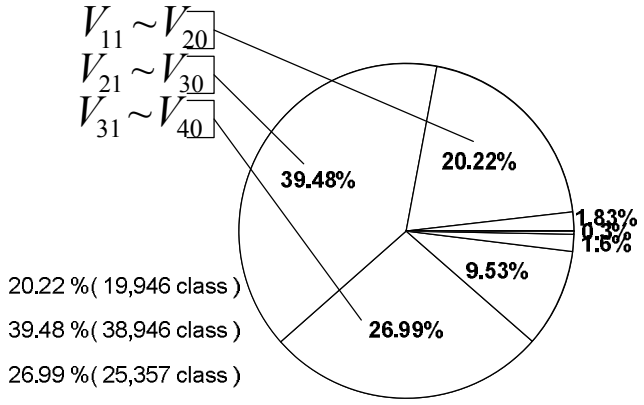

그림 2. number of nth parameter HMM

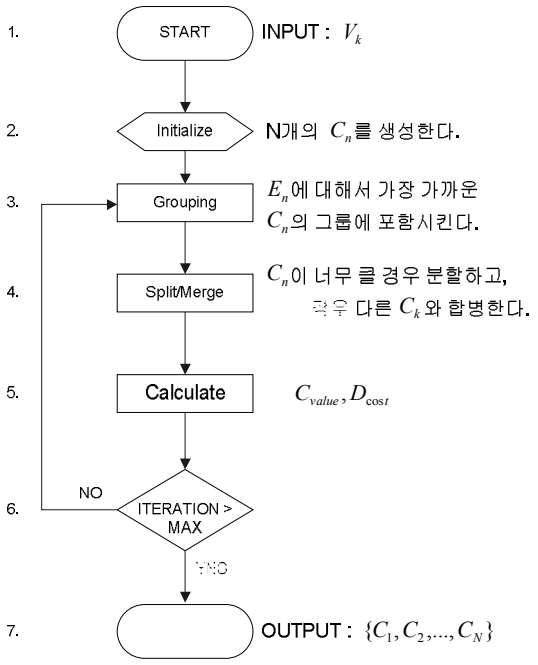

그림 3.

단계 1 클러스터링의 진행과정은 그림 3과 같다. Initialize 단계에서 초기의 C_n을 배정하는 방법은 V_{pn}의 입력 집합 E_N을 한자 획 수 기준으로 정렬한 뒤 100개당 하나의 꼴로 C을 생성하였으며 V_{pn}에 존재하는 모든 획수 S에 대해서 최소한 1개의 C가 생성되는걸 보장하였다. 이 단계에서 생성되는 초기 C의 개수는 평균적으로 $\frac{K}{init:100}, K \text{ is } \{E_1,...E_K\} \in V_{pn}$ 와 같다.

Grouping 단계에서는 가장 우선적으로 E_n이 C_i에 포함될 수 있는 전제 조건을 만족하는지를 확인한다. 그 전제조건이란 $C_{i\ value}$의 한자 획 수에 대하여 E_n의 한자 획수 차이가 +-20%이내 이고, 구조 코드 열의 길이 또한 마찬가지로 +-20%이내의 차이 이어야 한다. 이는 구조 코드열의 길이와 한자 획 수의 길이의 차이가 일정 한도 이상 벗어날 경우 두 한자간의 유사성은 기대하기 힘들다는데 비롯된 전제조건이다. 전제조건을 만족하는 E_n와 C_i간에 distance 계산에 사용되는 C_{value}의 정의와 distance 계산방법은 식 2과 같다.

$$C_{value} = \frac{\sum_{k=1}^{k} E_k value}{k}, E_k \in C$$

$$DISTANCE(C_{value}, E_n) ::$$

FOR i TO number of non - null transition

  FOR j TO number of observation sequence

    Distance + = absolute(C transition $prob_i$ * C observation $prob_{ij}$ −

    E transition $prob_i$ * E observation $prob_{ij}$)

식 2.

식 2는 C_{value}와 E_n에 대하여 모든 관측 코드로 인한 한 상태에서 다른 상태로 전이 될 상태전이 확률간의 차이를 구하여 절대치 취하므로 각 상태에서의 동일한 관측 열에 대한 출력 확률을 비교할 수 있다. 이는 한 상태에서의 다른 상태로의 전이는 전이확률에 의해 결정되는 것이기는 하지만 그 확률의 크기를 결정하는 것은 현 상태에서 관측되는 관측 코드 확률에 의해서 이다. 그러므로 이 식은 미지의 관측 코드 열 입력에 대해 두 모델이 얼마나 근사한 출력 확률을 갖는지 확인하는 타당한 방법이다.

단, 식 2의 계산법은 두 모델간의 non-null transition의 개수가 동일해야만 한다. 다시 말해 non-null transition은 모델의 상태 수에 정비례 하므로 모든 클러스터링의 대상은 상태수가 동일한 모델들의 집합이라는 본 논문의 전제조건을 만족할 때만 성립한다. 이러한 전제조건이 없다면 서로 다른 상태수를 갖는 모델에 대해서 distance를 계산하기 위해서는 DP-matching 등과 같은 계산 복잡도가 높은 알고리즘을 사용해야 하기 때문에 시간이 오래 걸리고 구현 또한 좀 더 복잡해 진다.

식 3는 V_{pn}에서의 클러스터링이 얼마나 잘 되었는지를 판단하기 위한 척도로서 사용한 D_{cost}에 대해 정의한 것으로서 각각의 클러스터에 대하여 C_{value}와 클러스터의 원소간의 distance 값의 평균을 취하고 이렇게 구해진 값들의 총 합을 클러스터의 개수만큼으로 나누어 클러스터에 존재하는 원소들이 평균적으로 얼마나 근사한 원소끼리 뭉쳐있는지를 판단한다.

$$\mathrm{D}_{\cos t} = \min(D_{old}, \sum_{i=1}^{I}(\frac{\sum_{j=1}^{J} DISTANCE(C_{i\,th\,value}, E_j)}{J})/I)$$

식 3.

그림 3의 분할/합병(Split/Merge)단계는 ITERATION이 갖는 상수값에 따라 다른 방식으로 이뤄진다. 이 상수값에 따른 개별적 알고리즘을 설명하기에 앞서 ITERATION이 갖는 의미를 좀더 명확히 하자면 클러스터링 진행 단계 3~6까지의 반복 횟수를 뜻 하는 것이며, 최대 증가 상한 값 MAX까지 증가한다. 본 논문에서는 MAX의 값은 200으로 하였고 ITERARTION이 갖는 1~MAX까지의 범위의 값에 따라서 다른 분할/합병 알고리즘을 접목시킴으로 보다 좋은 클러스터를 얻을 수 있었다. 1~MAX값 사이를 갖는 범위는 크게 3부분으로 나뉜다.

- $RANGE_A$: 1~80
- $RANGE_B$: 81~Q
- $RANGE_C$: Q~200
- Q는 82~199 사이의 임의의 상수이다.

$RANGE_A$는 가장 평범한 단계이다. 이 단계에서 합병은 클러스터의 원소개수가 C_{min_number} : '80' 미만인 클러스터들을 대상으로 하나의 클러스터를 다른 클러스터와 합병 시켰다. 합병 할 수 있는 클러스터가 많을 경우 두 클러스터 간의 distance을 계산하여 가장 작은 값을 보이는 클러스터와 합병하였다. split과정은 클러스터의 원소 개수가 C_{max_number} : 120이 넘을 경우에 분할하였다. 분할 방법은 C에 포함된 서로 다른 모든 E에 대해서 자신과 가장 distance가 작은 E에게 투표를 하고, 투표율이 높은 순서로 ($size\ of\ C\ /\ C_{max\_number}$) +1개의 새로운 클러스터 C로 분할하였다. $RANGE_B$는 현재까지 갱신된 D_{cost}의 값이 다시 갱신될 때까지 계속적으로 클러스터링 과정을 진행하고, D_{cost}의 값이 더 작은 값으로 갱신이 되는 시점 'Q' 상태의 C_N의 총 개수, 즉 클러스터의 개수가 가장 좋다고 판단하여 더 이상 그림 3의 4번째 split/merge 단계를 통해 C_N의 개수를 변화하지 시키지 않는다. 이후 $RANGE_C$ 과정을 통해 'Q'~200번까지의 반복을 통해 고정된 C_N의 개수로 클러스터링을 진행하여 단계 1. Clustering을 종료한다. 본 논문에서 모든 V_{pn}에 대하여 생성한 C_N의 개수는 1,220개이다.

3.2 단계 2 클러스터링

이 단계는 전체적으로 본 논문의 클러스터링은 Top-

Down형태의 알고리즘을 취하고 있음을 보여준다. 단계 2 클러스터링은 단계 1 클러스터링과 전반적으로는 비슷하나 다음과 같은 세 가지 차이점이 있다. 첫 번째로 입력 데이터는 단계 1 클러스터링에서의 출력 데이터인 $C_N=\{C_1,C_2,\cdots,C_N\}$이고 이를 사용한 단계 2 클러스터링의 출력 데이터는 $C'_N=\{C'_1,C'_2,\cdots,C'_N\}$이다. C'_n은 C_n에 포함된 모든 E에 대해서 C_n보다 더 작게 클러스터가 구성된 집합이다. 즉, 단계 2. Clustering은 V_{pn}의 모든 원소 E을 클러스터링의 입력 데이터로 구성되는 C_n와는 다르게 V_{pn}의 원소 E들로 만들어진 C_n을 입력 데이터로 하므로 E의 전체 집합을 볼 때 Top-Down형식이다.

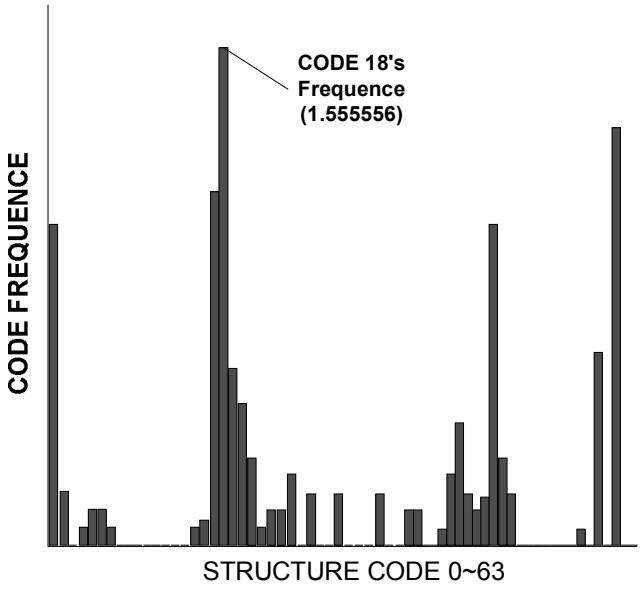

그림 4. '光'에 해당되는 모델의 Structure Code Histogram

두 번째는 단계 1 클러스터링에서 사용된 distance 계산식은 동일하지만 입력 데이터 원소 E_n이 C'_n에 포함될 수 있는 전제 조건에 각자의 모델을 훈련할 때 사용한 다수의 입력 데이터인 구조 코드 열이 가지고 있는 64개의 구조 코드들의 평균 값을 취한 '구조 코드 분포도'의 차이를 비교하는 기준을 추가하여 클러스터를 이루는 E_n들 간의 유사성 판단을 강화하였다. 여기서 구조 코드 열 평균 분포도의 차이를 비교한다는 것은 두 모델간의 구조 코드 열 평균 분포도의 차이를 절대값을 취해 합한 것을 말한다. 본 논문에서는 두 모델간의 구조 코드 분포도 차이가 30%이상일 경우 한 클러스터에 포함되지 않게 함으로서 더 좋은 인식률을 구현 할 수 있었다. 단계 1 클러스터링에서 이 계산식이 들어가지 않은 이유로는 구조 코드 열 평균 분포도를 사용한 distance계산법은 원소의 개수가 단계 1 클러스터링과 같이 많을 시에는 분별력이 떨어져 좋은 성능을 보이지 않기 때문에 단계 1 클러스터링에 비해 상대적으로 클러스터링을 할 원소의 개수가 적은 단계 2 클러스터링에서만 적용한 것이다.

세 번째는 Initialize 단계에서 생성하는 초기 C'_n의 개수를 정하는 상수가 *init*의 값이 '100'이 아닌 '25'이고, split/merge를 결정하는 C'_{min_number}의 값은 '20'이며 C'_{max_number}는 '25'이다. 이상 세 가지 차이점을 제외하고 단계 2. Clustering은 단계 1. Clustering의 나머지 구성을 따른다. 본 논문에서 단계 2. Clustering에 사용되는 모든 입력 데이터에 대해 생성된 출력 데이터 C'_n의 개수는 5,432이다. 단계 1과 2를 거쳐 최종적으로 만들어진 클러스터의 구조는 그림 5와 같다.

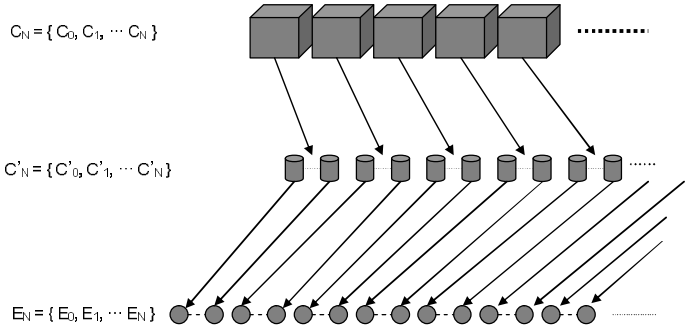

그림 5. 클러스터 모델구조.

4. 인식방법

4.1 모델 필터링

완성된 클러스터 모델의 구조에서 필기 한자 인식을 하는 과정을 간략하게 설명하면 모든 C 에 포함된 E 중에 $INPUT_{DATA}$(미지의 입력 데이터)의 모델에 해당 될 것이라 기대되는 E 을 찾고, 구해진 E 들을 $INPUT_{DATA}$에 대해서 출력되는 확률이 높은 순서로 사용자에게 보여주는 것이다.

본 논문에서 제안한 클러스터링의 전제조건 중 하나의 C는 C_{value}의 한자 획 수 및 코드 열 길이와의 차이가 20%미만인 E들로 구성 한다는 것이 있었다. 그렇기 때문에 $INPUT_{DATA}$와 설명한 전제조건을 만족하지 않는 C 에 대해서는 Distance을 구하지 않으므로 $INPUT_{DATA}$로 찾고자 하는 모델일 가능성이 없는 클러스터를 쉽게 제외 시킬 수 있다. 이 과정을 모델 필터링이라 한다. 모델 필터링은 인식과정 중에 총 3번 이뤄진다.

4.2 인식과정

본 논문의 클러스터 모델구조는 하나의 $INPUT_{DATA}$를 인식하기 위해서는 아래와 같은 단계에 걸쳐 출력 확률을 측정해야 한다. 각 단계에서 필요로 하는 출력 확률의 측정은 Forward Pass 알고리즘을 사용하여 구할 수 있다.

1) 모델 필터링을 통과한 C 들에 대하여 $INPUT_{DATA}$에 대한 출력 확률을 구하고, 높은 확률을 보이는 순서로 정렬한다.

2) 정렬된 결과물 C 들에 포함되어 있는 C' 중에 모델 필터링을 통과한 대상에 대하여 $INPUT_{DATA}$에 대한

출력 확률을 구하고, 마찬가지로 높은 확률 순서로 정렬한다.

3) 구해진 C' 에 포함되어 있는 E중에 모델 필터링을 통과한 대상에 대하여 $INPUT_{DATA}$에 대한 출력 확률을 구하고, 최종적으로 출력 확률이 높은 순서로 정렬하여 인식결과를 사용자에게 알려준다.

각 단계에서 출력 확률 계산의 대상이 되는 C', C, E의 관계는 $E \in C' \in C$ 와 같다. 즉, 3) 단계에서 인식대상이 되는 E의 개수는 궁극적으로 모델 필터링을 통과한 C 의 개수에 의존된다. 그러나 1) 단계에서 필터링을 통과한 C 의 개수에 항상 정비례 하지는 않는다. 또 모델 필터링을 통과한 모든 C 가 다음 단계에서 계산할 C' 들의 집합은 아니다. 그 이유는 다음과 같다.

모델 필터링을 통과한 C 에 대해서 $INPUT_{DATA}$에 대한 출력 확률이 높은 순으로 정렬하여 상위 X개만을 다음 단계에서 계산할 C' 들의 집합으로 활용한다. 이는 C_{value}가 C 에 포함하는 E들이 갖는 HMM 모델링 능력에 비해서는 약하지만 자신의 원소에 해당되는 E가 표현하는 실제 클래스에 대해서 어느 정도의 HMM 모델링 능력을 보여줄 것이라고 가정했기 때문이다. 이것을 가정 함으로서 모델 필터링 하나만을 적용하여 인식 대상이 되는 모델을 제외시키는 방식에 비해 비약적으로 많은 인식 대상이 되는 모델의 수를 줄일 수 있다. 하지만 이 방법에는 해결해야 할 문제가 있다. 구체적으로 상수 X를 어떻게 정의하느냐이다. 상수 X는 C_{value}, C'_{value}가 자신의 원소로 포함되어 있는 모든 E의 실제 클래스들을 얼마나 잘 표현하는 모델인가에 따라서 달라진다. 이 것은 항상 일정한 것이 아니라 클러스터링을 통해 모델이 어떻게 생성되었는지에 의존적이다. 그러므로 C_{value}, C'_{value} 가 얼마나 좋은 값을 가지고 있는지 평가를 하여 X값을 구하였다. C_{value}, C'_{value}의 평가 방법은 대량의 $INPUT_{DATA}$에 대해서 C, C'를 C_{value}, C'_{value}의 출력 확률로 정렬하고, 이때 $INPUT_{DATA}$에 해당되는 실제 모델 E가 포함된 C, C' 가 몇 순위에 있는지를 실험하여 $INPUT_{DATA}$의 코드 열 길이를 기준으로 평균값을 구한다. 이렇게 측정된 상수 값 X는 코드 열 길이 n을 갖는 $INPUT_{DATA}$에 대한 실제 모델 E를 적중 시킬 C', C 의 모델링 능력이라 정의한다. 본 논문에서는 실제로 X의 값을 적용할 때는 코드 열의 길이 별로 적용시키지 않고, 10의 단위로 나누어 범위를 정하여 적용 하였다.

표 2는 본 논문에서 사용한 클러스터 모델을 평가하여 구한 C_N에 대해 적용된 X의 값을 보여준다.

표 2.

X_n	Code Number	Value
X_0	2~9	100
X_1	10~19	550
X_2	20~29	550
X_3	30~39	350
X_4	40~49	250
X_5	50~59	202
X_6	60~69	50
X_7	70~	4

4.3 구현

98,639 종류의 한자에 대한 HMM을 모두 시스템 메모리상에 상주 시키기에는 메인메모리가 4GB가 장착된 PC에서도 가능하지 않을 정도로 많은 양이다. 그렇기 구현한 한자 인식기는 98,639에 해당하는 E 를 메모리에 상주시키지 않고 6652개의 C, C' 을 표현하는 정보만을 메모리에 상주 시키고, E 에 해당되는 HMM정보를 HDD에 저장한다. 비록 C 나 C' 가 자신의 클러스터에 속하는 원소들의 HDD상의 위치와 같은 정보를 알고 있어야 하므로 E 에 비해서 유지해야 하는 정보의 양은 조금 많지만 C, C', E 을 실질적으로 표현하는 대부분의 정보의 양은 HMM을 이라는 것을 고려해보면 C, C'와 E 간의 정보량의 차이는 무시 할 정도라는 것을 알 수 있다.

실험을 위해 구현한 한자 인식기는 모든 E 에 대한 정보를 메모리에 상주 시키는 방법에 비해 약 1/14 가량의 메모리만을 사용한다. 또 하나의 $INPUT_{data}$를 인식하기 위한 E 에 대한 정보는 필요할 시에 필요한 만큼만 HDD에서 읽어와 메모리상에서 사용한다. 여기서 "필요할 시 필요한 만큼" 이란 말은 $INPUT_{DATA}$를 인식하기 위해 4.2절의 인식과정 단계 2)에서 구해진 모든 C' 에 대해 $INPUT_{DATA}$와의 출력 확률이 높은 순서로 정렬이 되고 나면 X_n개 만큼의 C'의 원소 E 들에 대하여 출력 확률을 계산하여야 하는데 이때 필요한 E 들에 대한 HMM 확률 정보를 C''에 포함된 E 의 집합 단위로 HDD에서 메모리로 읽어와서 마찬가지의 단위로 $INPUT_{DATA}$와의 출력 확률을 계산하고 계산이 완료된 E 에 대한 정보는 메모리 상에서 삭제한다. 여기서 구체적으로 E 를 HDD에 기록해두는 방식은 V_{pn} 단위로 하나의 파일을 구성하고 있으며, 각 파일은 첫 부분은 V_{pn}에 해당되는 C'_0에서 C'_N까지의 클러스터 단위로 E 의 확률 정보가 기록되어 있다. 본 논문에서 구현한 한자 인식기의 시스템상의 메모리 점유율은 249메가이고, 모든 E의 HMM 정보가 기록된 파일의 사이즈는 1,348메가 이다.

5. 실험 및 결과 분석

본 논문에서 98,639 종류의 한자를 인식하기 위해 남녀 44명에게 98,639자 * 27벌, 총 2,663,253자를(1벌의 단위는 98,639자) 수집하였다. 이중에서 20벌에 해당되는 1,972,780자는 HMM을 훈련하는데 사용하였고, 나머지 7벌에 해당되는 690,373자는 인식률 측정을 위해서 20벌로 훈련된 HMM으로 구성된 한자 인식기의 입력 테스트 데이터로

사용하였다. 인식 결과는 3절 클러스터링에서 구조 코드 열 히스토그램의 사용 여하에 따른 1~10위, 1~20위, 1~30위, 1~50위까지의 인식후보 한자들에 대해서 인식률과 문자당 인식속도를 측정하였다. 표 3은 테스트 데이터 680,373개를 사용하여 측정한 결과이다.

표 3.

N순위후보 / 인식률	10위	20위	30위
SC_Histogram 미적용	90.06	93.22	94.64
SC_Histogram 적용	92.21	94.57	96.03
인식시간	920(ms/char)		
# of training data = 1,972,780 (char) # of test data = 690,373 (char)			

6. 결론

본 논문에서 대상이 된 한자 종류는 98,639자로서 HMM을 사용한 일반적인 한자 인식방법으로는 좋은 인식시간을 얻기 어렵고, 시스템 리소스의 제약으로 인해 실용화 역시 어렵다. 본 논문에서는 클러스터링을 사용하여 이 문제를 극복하였다. 효과적인 클러스터링을 위해 동일한 상태 수를 갖는 HMM을 대상으로 클러스터링을 적용했다. 이미 훈련된 HMM 파라미터에 근거하여 거리 계산을 함으로써 비슷한 모델들이 동일한 클러스터에 자연스럽게 포함될 수 있도록 하였다. 또는 구조 코드 히스토그램을 추가로 사용함으로써 클러스터링의 성능을 높일 수 있었다. 제안한 방법은 한자의 획 수, 생김새 등과 같은 사전 관찰을 통해 비슷한 한자를 하나의 클러스터로 분류를 하는 방법을 사용하지 않음으로써 입력 데이터의 길이가 고정되지 않은 온라인 필기 데이터를 효과적으로 클러스터링할 수 있었다.

본 논문에서는 약 10만자의 한자에 대해 1초 이내에 인식결과를 얻을 수 있었고, 그 때의 메인메모리 사용량은 250MB이내였다. 하지만 보다 실용적인 시스템을 위해서는 인식속도와 인식률 향상이 더 요구된다.

참 고 문 헌

[1] 김형태, 하진영, “온라인 필기 한자인식을 위한 체인코드열과 구조코드열의 성능평가,” 한국정보과학회 2006 가을 학술발표논문집(B), 한국정보과학회, pp.402-407, 2006.

[2] J.-Y. Ha, “Structure Code for Hmm Network-based Hangul Recognition," 18th International Conference on Computer Processing of Oriental Language, ICCPOL 99, pp.106-113, 1999

[3] 하진영, 신봉기, “온라인 한글 인식을 위한 HMM 상태 수의 최적화”, 한국정보과학회 Vol. 25, No. 2, 1998.

[4] 하진영, “HMM 네트워크 기반의 한글 인식기를 위한 구조 특성열의 적용,” 제 10회 한글 및 한국어 정보처리 학술대회, 1998.

[5] R.O. Duda, P.E. Hart, and D.G. Stork, Pattern Classification (2nd ed), John Wiley & Sons, Inc., 2001.