Journal of The Korea Institute of Information Security & Cryptology(JKIISC) ISSN 1598-3986(Print)
VOL.23, NO.5, October 2013 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2013.23.5.909

효과적인 글자 분리 방법을 사용한 네이버 캡차 공격*

양 대 헌,[1†] 최 용 헌,[1] 홍 석 준,[2] 이 경 희[3‡]
[1]인하대학교 컴퓨터정보공학과, [2]인하대학교 정보통신공학과, [3]수원대학교 전기공학과

Analysis of Naver CAPTCHA with Effective Segmentation*

DaeHun Nyang,[1†] YongHeon Choi,[1] SeokJun Hong,[2] Kyunghee Lee[3‡]
[1]인하대학교 컴퓨터정보공학과, [2]인하대학교 정보통신공학과, [3]수원대학교 전기공학과

요 약

웹 서비스를 이용하는 대상이 사람인지 아닌지를 구분하기 위해 주로 사용되는 캡차는 초기 단순한 문자열을 보고 입력하는 것을 시작으로 계속하여 발전해왔다. 지금까지 다양한 종류의 캡차가 제안되었으나 아직까지는 대부분의 사이트에서 편의상 간단한 문자 기반의 캡차가 널리 쓰이고 있다. 이 논문에서는 한국에서 검색을 하는 사용자 중 70% 이상이 이용하는 네이버에서 사용하고 있는 캡차를 새로운 글자 분리 방법을 이용해 공격함으로써 해당 캡차의 취약성을 검증해 보았다. 실험 결과, 총 1000번 중 938번을 성공해서, 네이버의 캡차 방식이 논문에서 제시된 공격에 의해 안전하지 않음을 입증하였다.

ABSTRACT

CAPTCHA is an automated test to tell apart computers from human mainly for web services, and it has been evolved since the most naive form in which users are requested to input simple strings has been introduced. Though many types of CAPTCHAs have been proposed, text-based CAPTCHAs have been widely prevailed for user convenience. In this paper, we introduce new segmentation schemes and show an attack method to break the CAPTCHA of Naver that occupies more than 70% of the market share in search engine. The experimental results show that 938 trials out of 1000 have successfully analyzed, which implies that we cannot use the CAPTCHA anymore.

Keywords: CAPTCHA, Breaking CAPTCHA, SVM, Character Separation

I. 서 론

캡차(Completely Automated Public Turing test to tell Computers and Humans Apart)는 대부분의 사람이 풀 수 있지만 컴퓨터 프로그램은 풀 수 없는 문제를 생성하고 평가할 수 있는 프로그램이다.[1,2] 주로 프로그램에 의해 웹 서비스에 대해 반복적인 동작을 수행하는 것을 방지하기 위해 사용하는데, 서비스의 대표적인 예로는 온라인 투표, 무료 이메일 서비스, 검색 엔진 봇 차단, 이메일 웜 및 스팸 차단, 사전 공격(Dictionary Attack) 차단이 있다.

현재 대부분의 캡차는 무작위의 문자열을 보여주고 이를 맞추도록 하는 형태를 취하고 있는데, 이 논문에서 대상으로 하는 네이버의 캡차 또한 같은 형태로 이루어져 있다. 네이버의 캡차는 이미 한차례 김성호 등에 의해 공격 당해 컬러정보를 제거한 새로운 캡차로 변경되어 서비스 되고 있다[7]. 이 논문에서는 이미지 처리를 통해 문자를 분리하는 알고리즘과 SVM(Support Vector Machine)을 이용하여 현재 서비스 되고 있는 네이버 캡차를 인식하고자 하였다. 이 논문에서 제시한 알고리즘은 다음의 세 가지 collapsing

접수일(2013년 6월 10일), 수정일(2013년 7월 23일), 게재확정일(2013년 8월 14일)
* 이 논문은 인하대학교의 지원에 의하여 연구되었습니다.
† 주저자, nyang@inha.ac.kr
‡ 교신저자, khlee@suwon.ac.kr(Corresponding author)

을 해결했다는데 의의가 있으며, 다른 캡차를 공격하는데 구성요소로 사용할 수 있을 것이다:(1) 임의의 각도로 글자를 기울게 하는 Tilting, (2) 글자 사이의 간격을 없애는 방법, (3) 글자들의 폭이 고르지 않게 하는 방법. Ⅱ장에서는 문자열 기반 캡차와 이를 인식하기 위해 사용한 SVM에 대한 소개, 그리고 이 논문의 공헌에 대해 언급하고, Ⅲ장에서는 네이버 캡차의 문자를 분리하기 위한 이미지 처리 알고리즘에 대해 서술하였다. Ⅳ장에서는 실험 환경 및 결과에 대한 분석을 서술하고, Ⅴ장에서 이 논문을 마무리한다.

II. 기반 지식

2.1 문자열 기반 캡차

문자열 기반 캡차는 일반적으로 임의의 왜곡된 문자열을 노이즈와 함께 이미지의 형태로 제공하여 이를 맞추는 형태를 취하고 있다. 시각장애인의 이용이 어려워 웹 접근성에 위배되고 다른 종류의 캡차에 비해 상대적으로 프로그램에 의해 인식될 가능성이 높다는 단점을 가지고 있으나, 구현의 간단함으로 인하여 많은 웹 사이트에서 문자열 기반 캡차를 사용하고 있는데, 대표적으로는 캡차 문자열과 OCR로 인식하지 못하는 문자열을 같이 맞춰야 하는 reCAPTCHA가 있다[6]. 이 논문에서 대상으로 하는 네이버의 캡차 또한 여섯 개의 글자가 왜곡된 형태로, 일반적인 문자열 기반 캡차의 특징을 지니고 있다.

[그림 1] 대표적인 문자열 기반 캡차인 reCAPTCHA[6]

2.2 SVM

SVM은 기계학습분야에서 널리 사용되는 분류 알고리즘의 한 종류로, 선행연구에서 문자 인식을 위해 SVM을 사용하여 양호한 결과를 낸 선례가 있어 채택을 하였다. 이 논문에서는 네이버 캡차의 왜곡된 문자열을 일정한 알고리즘에 따라 문자 단위로 잘라낸 후, SVM을 통해 학습시켜 프로그램에 의해 인식이 가능함을 보일 것이다.

2.3 Collapsing 해결을 위한 접근 방법

캡챠를 공격하기 위한 많은 연구가 있었는데, 그 중 가장 최신의 연구 결과는 2011년 Bursztein 등이 이들을 모두 정리해서 실제로 구현한 것이다[10]. 이 논문에서는 캡챠를 공격하기 위한 일반적인 단계와 각 단계에서의 기법들을 제시했다. 이 논문에서도 언급되고 있는 바와 같이 가장 중요한 단계는 글자 분리(Segmentation), 그리고 글자 인식 (Recognition)인데, 글자 인식을 위해서는 SVM (Support Vector Machine), KNN (K Nearest Neighbors) 등의 범용적인 분류기를 제시했고, 글자 분리를 위해서는 CFS (Color Filling Segmentation) 이라는 방법만을 제시했다[11]. 글자 인식이 사실상 SVM과 KNN 등의 범용적인 분류기에 의존할 수밖에 없다는 점을 고려하면, 결국 캡챠를 공격하는 기법에서 가장 의미 있는 부분은 글자 분리 기법이라고 할 수 있다. [10]에서 anti-segmentation 기법으로 추천한 것은 collapsing으로 글자의 분리를 매우 어렵게 한다. Collapsing은 글자 사이의 간격을 없애거나, 글자들을 충분히 기울이게 해서 (Tilting) 글자 분리를 어렵게 하는 기법이다. 네이버나 reCAPTCHA의 경우 글자를 임의의 각도로 휘어지게 해서 분리를 어렵게 하고 있으므로, Collapsing이 적용되었다고 볼 수 있다. 따라서, 이에 대한 새로운 접근 방법이 필요하다. 이 논문에서는 이 tilting 기반의 collapsing을 해결하기 위해, 휘어진 방향에 따라 캡차를 두 가지 타입으로 구분했고 이를 각기 다르게 처리했다. 두 번째로, 네이버 캡차의 경우 가운데 두 글자는 다른 글자에 비해 폭이 좁고, 붙어있는 경우가 많은데, 이는 Collapsing 기법의 하나로 글자의 분리를 더욱 어렵게 만든다. 이를 해결하기 위해 가운데 붙어있는 두 글자를 분리하는 아이디어를 제시했다. reCAPTCHA 등 글자를 임의의 곡선 형태로 배열하는 기법이나 붙어있는 글자를 분리하는 데에도 적용할 수 있으리라 기대한다. 따라서, 이 논문의 아이디어는 캡챠 공격 기법, 좀 더 구체적으로는 캡챠 공격을 위한 anti-segmentation 기법 중 다음의 collapsing 기법을 해결 하는데 공헌이 있다고 할 수 있다. (1) 임의의 각도로 글자를 기울게 하는 Tilting, (2) 글자 사이의 간격을 없애는 방법, (3) 글자들의 폭이 고르지 않게 하는 방법. 이 세 가지 collapsing 기법은 흔히 쓰이는 anti-segmentation 기법으로, 이 논문에서 제시한 아이디어를 이용해서 위의 기법들을 사용

하는 다른 캡차를 공격하는데 활용할 수 있으리라 기대한다.

III. 캡차 분석 알고리즘

3.1 캡차 대상의 특징

네이버 카페 가입, 로그인 실패 시 사용되는 캡차는 [그림 2]와 같은 형태로 이루어져 있다. 캡차 인식을 위해 분석한 특징을 정리하면 아래와 같다.

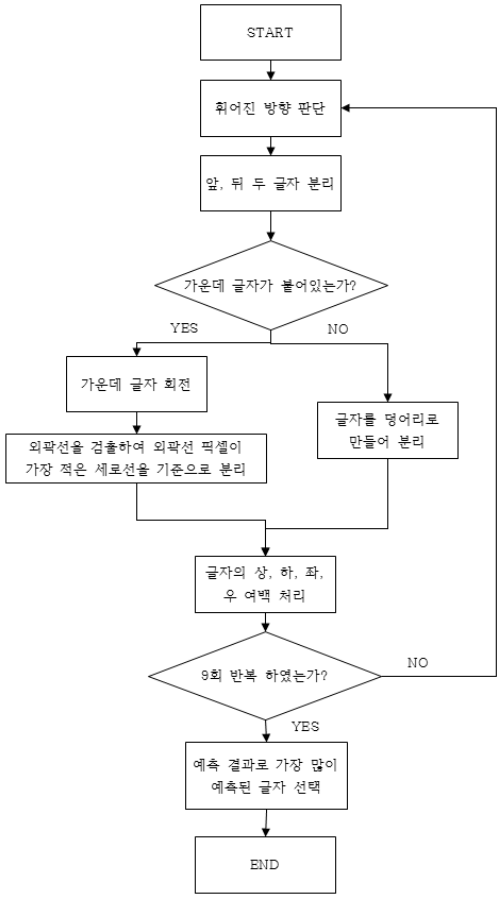
[그림 2]가입 시도 시 제시되는 캡차 이미지 예시

- 각 캡차 이미지는 6개의 문자로 이루어져 있다.
- 일정 시간 내에는 이미지가 왜곡된 정도만 달라지고 이미지의 문자가 바뀌지는 않는다.
- 가입을 다시 시도할 경우 문자가 바뀐다.
- 글자의 휘어진 방향이 2가지 방향이다.
- 앞, 뒤 글자의 간격에 큰 차이가 없다.
- 가운데 글자의 경우 회전된 각도에 큰 차이가 없다.

이러한 특징을 이용하여 6개 글자를 분리하였다.

3.2 흐름도

캡차 이미지를 문자로 인식하기 위해 [그림 3]의 흐름도의 내용으로 진행하였다. 분리 과정에 앞서, 네이버 카페 가입 페이지의 HTML코드를 분석하여 해당 캡차가 캡차 링크[9]에 몇 가지 파라미터를 넘긴 결과를 표시하는 것을 확인하였다. 이 중 암호화된 것으로 추정되는 'key'라는 필드의 파라미터를 통해 캡차의 문자열이 결정되는 것으로 추정하였고, 이 'key' 필드 값을 임의로 입력하거나 입력하지 않는 경우, 일정 시간을 초과한 후 캡차를 다시 요청하는 경우 임의의 캡차가 생성되는 모습을 보였다. 캡차를 인식하기 위해 가입 요청 과정에서 캡차가 동일하게 유지되는 동안 이미지를 가입 페이지에서 받아오게 되는데, 판독 성공률을 올리기 위해 글자는 같지만 형태가 다른 캡차 이미지 9장을 다운로드한 후 과반수투표제를 적용하였다. 다운로드한 이미지를 대상으로 분리작업을 진행한 후 사전에 학습해놓은 모델을 기반으로 하여



[그림 3] 캡차 인식 과정 흐름도

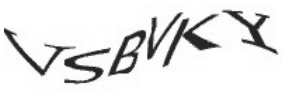
잘라낸 이미지를 각 글자별로 SVM을 이용하여 예측을 한다. 각 글자별로 예측 결과가 가장 많이 나온 문자를 해당 글자의 결과로 선택하여 전체 문자열의 결과를 완성한다.

3.3 분리 과정

3.3.1 휘어진 방향 탐색

본 캡차 이미지의 휘어진 방향은 [그림 4]과 [그림 5]와 같은 두 가지 형태로 정해져 있다. 어떤 방향으로 휘어져 있는지 찾기 위해, 이미지의 윗부분 또는 아랫부분을 일정 부분 잘라내어 어느 방향에 픽셀이 치우쳐 있는지를 판단하였다. 만약 문자열의 좌측이 [그림 6]과 같이 위로 올라간 물결무늬라면 가로축을 기준으로 일정 부분 잘라내었을 때 잘라낸 이미지의 왼쪽부분에 색이 있는 픽셀이 치우쳐 있을 것이다.

〔그림 4〕좌측 위로 올라간 무늬



〔그림 5〕우측 위로 올라간 무늬

〔그림 6〕 캡차 이미지에서 세로축으로 일정부분 잘라낸 결과

문자열의 우측이 위로 올라가있다면 반대로 잘라낸 이미지의 오른쪽 부분에 색이 있는 픽셀이 치우쳐 있을 것이다. 이러한 방식으로 35개의 이미지에 대해 테스트해 본 결과 35개의 이미지 전부에서 휘어진 방향을 올바르게 인식하였다.

〔그림 7〕은 휘어진 방향을 인식하는 알고리즘을 수도코드 형태로 나타낸 것이다. 수도코드 안의 변수 right_top과 left_top은 캡차 이미지 가로축의 가운데를 기준으로, 각각 오른쪽 윗부분에 존재하는 글자 픽셀 수와 왼쪽 윗부분에 존재하는 글자 픽셀수이다. 각 픽셀수를 체크하여 픽셀 수가 더 많은 쪽으로 휘어진 방향을 결정한다.

```
int function image_type()
right_top = 0
left_top = 0
for pixel px from left top to right bottom
    if px is not white
        if current x-axis position of x < 100
            left_top++
        else
            right_top++
        ENDIF
    ENDIF
ENDFOR
return (left_top > right_top)?left_top :
right_top
```

〔그림 7〕 휘어진 방향 인식 알고리즘

3.3.2 앞, 뒤 글자의 분리

캡차 이미지 50개를 직접 분석한 결과 휘어진 방향에 무관하게 앞, 뒤 글자의 간격의 큰 변동이 없었다.

실제로 〔표 1〕의 결과 값을 이용하여 평균에 가까운 값을 특정 크기로 정하여 이미지의 앞, 뒤 글자를 분리 해 본 결과 좋은 성능을 나타내는 것을 볼 수 있었다. 따라서 일정크기만큼 정하여 앞, 뒤 글자를 분리하였다.

〔표 1〕 50개의 이미지 간격 측정 결과

	첫 번째	두 번째	다섯 번째	여섯 번째
평균 (pixel)	39.38	38.26	39.56	38.28
표준편차	3.42	5.07	4.71	3.22

3.3.3 가운데 글자의 분리

가운데 글자의 경우 앞, 뒤 글자와는 다르게 휘어진 정도가 좀 더 심하기 때문에 일정한 간격으로 잘라내는데 어려움이 있었다. 대신 가운데 글자의 경우 휘어진 정도가 대부분의 캡차 이미지 인스턴스에서 비슷하다는 점을 이용하여 분리하였다. 또한 좀 더 완벽한 분리를 위하여 가운데 글자가 붙어있는 경우와 떨어져 있는 경우를 따로 처리 하였다.

가운데 글자가 떨어져 있는지 붙어있는지를 판단할 근거로는, 앞뒤글자를 분리한 뒤의 나머지 이미지에서 각 덩어리를 이루는 픽셀의 개수를 세었다. 만약 한 글자의 픽셀 수로 나올 수 없는 개수 이상의 픽셀수가 덩어리를 이루고 있다면 붙어있다고 판단하여 처리를 진행 하였다.

3.3.3.1 붙어있다고 판단한 경우

첫째로 글자의 휘어진 방향에 따라 음의 각도 또는 양의 각도 크기만큼 회전시키는 작업을 하였다. 가운데 글자가 휘어져 있다 해도 대부분의 이미지 인스턴

〔그림 8〕 회전시킨 이미지

스의 가운데 글자의 휘어진 정도가 큰 차이가 없었기 때문에 위와 같은 방법을 사용할 수 있었다. 캡차를 회전시키면 〔그림 8〕과 같이 가운데 글자를 나누는 선이 세로축과 거의 평행하게 된다.

둘째로, 회전시킨 이미지를 대상으로 Canny Edge Detection Algorithm을 적용하여 외곽선을 검출하였다. Canny Edge Detection Algorithm은 John Canny가 개발한 외곽선 검출 알고리즘으로, 검출 과정이 다른 외곽선 검출 알고리즘에 비해 복잡한 편이지만 외곽선 검출 성능이 뛰어나 널리 이용되고 있어 이를 사용하였다.〔8〕 글자를 그대로 두지 않고 외곽선을 검출하는 과정을 추가한 이유는 글자의 굵기가 분리할 위치를 결정하는 데에 영향을 미치기 때문에, 외곽선 안의 부분을 배제하여 좀 더 정확한 분리를 할 수 있도록 하기 위함이다.

셋째로, 외곽선을 검출한 이미지를 대상으로 가운데 글자가 분리될 것이라고 예측한 특정 범위 내에서 세로 방향으로 픽셀을 세어 가장 픽셀이 적은 부분 즉, 세로축의 값에서 외곽선과 만나는 부분이 가장 적은 부분을 선택하여 그 부분을 분리해 내었다. 〔그림 8〕을 대상으로 분리 과정을 진행하였을 때 〔그림 9〕와 같은 결과를 나타낸다.

〔그림 9〕 회전된 이미지 분리

3.3.3.2 붙어있지 않다고 판단한 경우

각 글자를 덩어리로 만들어서 한 덩어리를 이루는 픽셀의 개수가 가장 많은 2개의 글자를 가운데 두 글자로 간주하고 분리시킨다. 단순히 픽셀의 개수에 따라 2개의 글자를 가져오기 때문에 어느 글자가 왼쪽의 글자인지, 오른쪽의 글자인지 알 수 없기 때문에 분리된 2개의 글자를 대상으로 어느 방향에 픽셀이 치우쳐 있는지를 검사하여 오른쪽의 글자인지 왼쪽의 글자인지를 판단한다. 또한 가장 큰 2개의 덩어리 이외의 덩어리는 글자가 아닌 파편으로 간주하고 제거한다.

〔그림 10〕 붙어있지 않은 경우의 이미지

아래 〔그림 10〕의 예시는 가운데 두 글자가 붙어있지 않는 경우이다.

〔그림 11〕은 가운데 두 글자를 분리하기 위해 먼저 두 글자가 있는 부분을 잘라낸 그림이다.

〔그림 11〕 붙어있지 않은 경우의 가운데 글자

잘라낸 가운데 두 글자를 이용하여 각 글자 'F'와 'A'를 덩어리로 만들어내어 분리한다. 결과는 아래 〔그림 12〕과 같다.

위의 방식으로 처리된 글자의 경우 이미 덩어리로 만들어서 가장 덩어리를 이루는 픽셀의 개수가 많은 두 덩어리를 분리해 낸 것이기 때문에 후 처리 과정에서 파편을 제거하는 과정을 거칠 필요가 없다.

〔그림 12〕 분리된 두 글자

3.4 후 처리 및 예측과정

3.4.1 회전된 이미지의 복구

가운데 글자를 분리해 내기 위하여 임시로 회전 시켰던 글자를 다시 원래대로 회전시킨다. 별로 의미 없는 작업으로 보일 수 있으나, 실제로 회전된 이미지를 복구하면서 쓸모없는 파편들이 제거되었으며 글자가 좀 더 정자체로 보이기 때문에 회전된 이미지를 복구하는 과정은 SVM을 통한 예측 판독률에 상당한 영향

〔그림 13〕 회전된 이미지의 복구결과

을 끼쳤다.

위의 〔그림 9〕에서 본 결과의 경우에도 회전된 이미지를 다시 원래대로 복구하였을 경우 아래의 〔그림 13〕와 같은 결과를 얻을 수 있었다. 아래의 그림에서도 알 수 있듯이 위의 〔그림 9〕에 있었던 파편들이 많이 사라진 것을 알 수 있다.

3.4.2 파편 제거

앞, 뒤 글자에 대하여 임의로 정해진 간격으로 분리를 하였기 때문에 자기 글자가 아닌 다른 글자의 일부가 함께 잘린 경우가 발생한다. 다른 글자의 일부를 제거하기 위하여 픽셀을 세어 작은 덩어리인 다른 글자의 일부를 제거하였다.

3.4.3 여백 제거 및 크기 조절

글자의 상하좌우에 있는 여백을 제거한 후, SVM의 정확도를 높이기 위해 크기를 40x40으로 조절하였다. 또한 SVM의 값은 0또는 1로 이루어지므로 최종 결과로 나온 이미지를 이진화 시킨다. 이 논문에서는 글씨가 있는 부분을 1, 배경 부분을 0으로 이진화 시켰다.

3.4.4 SVM 트레이닝

분리되어 완벽하게 SVM에 트레이닝 시키기 위한 세트로 만들어져 있는 이미지를 대상으로 SVM에 트레이닝을 하는 작업을 시행하였다. 트레이닝 대상은 2,000개로, 네이버 캡차 제공 링크〔9〕를 통해 다운로드 받은 후 분리하여 각 글자별로 입력 데이터를 만들었다. 입력 데이터를 만드는 과정에서도 글자의 인식률을 높이기 위한 방법이 있었다. 제대로 분리되지 않은 글자들의 패턴이 비슷하다는 점을 이용하였다. 예를 들어, 〔그림 14〕처럼 분리된 글자가 있을 경우 알파벳 'N'의 일부가 잘린 것으로 간주하고 입력 데이터에 포함하였다. 실제로 아래와 같이 분리된 글자의 대부분이 알파벳 'N'이었다.

〔그림 14〕 분리된 글자

위와 같은 유형의 알파벳은 'N'뿐만이 아니라 'H', 'D'등이 있었다.

입력 데이터를 작성한 후, 정확도가 어느 정도 되는지를 확인하기 위해 libsvm에서 제공하는 grid.py를 이용하였다. 그 결과 앞뒤 각 두 글자의 경우 높은 예측률을 보였으나 가운데 두 글자의 경우 예측률이 나머지에 비해 저조하여 1000개를 추가하여 가운데 두 글자는 3천개, 나머지 글자는 총 2천개의 캡차로 모델을 만들었다.

3.4.5 SVM 테스트

9개의 이미지를 대상으로 위의 분리 작업을 시행한 후 각 글자마다 트레이닝 되어 있는 모델 파일을 이용하여 예측을 시행하였다. 앞뒤 각 두 글자의 경우 grid.py에 의해 최적화된 파라미터를 이용하여 학습한 경우와, 기본 파라미터를 이용하여 학습한 경우의 차이가 거의 없었다. 반면, 가운데 두 글자의 경우 grid.py에서 최적화된 γ (RBF 커널의 계수) 파라미터를 학습과정에서 추가하지 않았을 때보다 추가하였을 때 예측률이 눈에 띄게 상승하는 결과를 볼 수 있었다. 각 자리별로 사용된 γ는 각각 0.000625, 0.000488281, 0.00195313, 0.0078125, 0.000625, 0.000625 이다. 각 결과는 임시로 저장해 두었다가 9개의 이미지의 예측이 모두 완료되면 각 글자별로 가장 많이 예측된 글자를 최종 결과로 결정하게 된다.

IV. 실험 결과

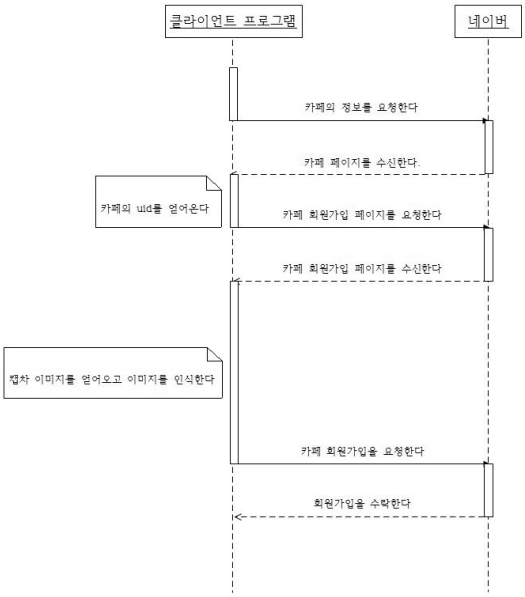
4.1 실험 조건환경

위의 설명한 내용을 바탕으로 실험을 하기위해 만든 프로그램과 실험을 진행한 컴퓨터의 환경은 아래와 같다.

〔표 2〕 실험 환경

개발 환경	Visual Studio 2010
사용 라이브러리	OpenCV 2.3.1〔4〕 libSVM 3.1.6〔5〕
사용 언어	C++
분류 알고리즘	SVM
테스트를 진행한 컴퓨터	Intel® Core™ i3 M380 @ 2.53GHz, 3GB

SVM 모델 생성 시 다운로드한 캡차 이미지와 별도로, 이번 실험에서는 카페 가입 단계를 1000번 반



〔그림 15〕 네이버 카페 가입 과정

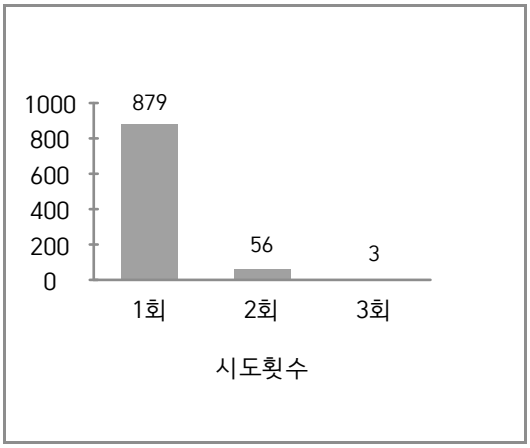
복하여 key 필드의 값이 유지되는 동안, 가입 시도마다 캡차 이미지를 9개씩 1000세트 다운로드하여 실제로 어느 정도 정확한지를 시험하였다. SVM을 이용하여 예측한 문자열과 사람이 직접 인식한 문자열을 비교하여 6글자 전부 일치하였을 때 성공으로, 한 글자라도 틀린 경우 실패로 간주하였다.

모델 파일 작성 시 사람이 알아볼 수 없게 잘린 경우 포함하지 않았으며, 제대로 잘리지는 않았으나 글자의 특성이 남아있는 경우에는 모델 파일에 포함하였다. 가입 과정의 전체적인 과정은 〔그림 15〕의 시퀀스 다이어그램과 같다.

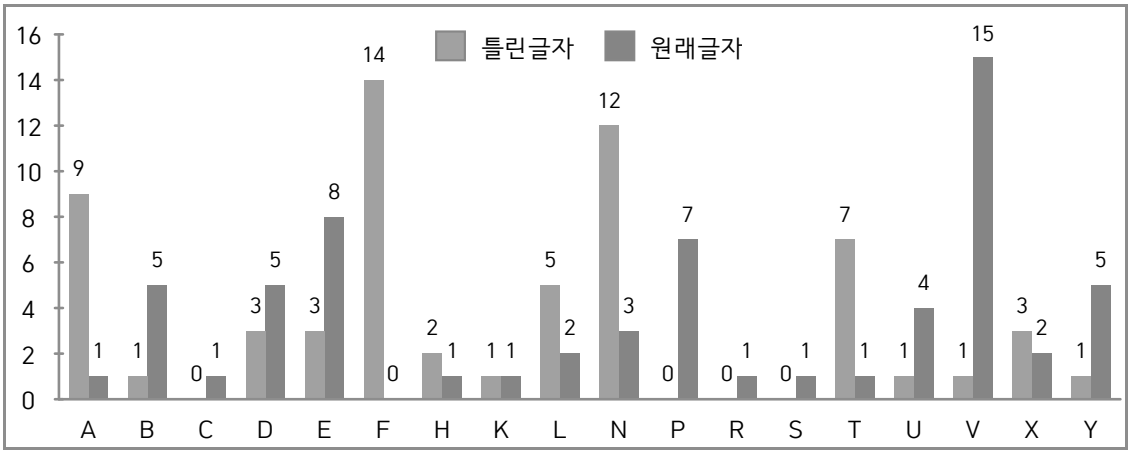
4.2 실험 결과

앞에서 제안한 방식과 실험 환경에 제시된 조건을 기초로 하여 캡차 이미지 인식을 시도하였고, 실험을 통해 얻은 데이터를 토대로 다음과 같은 결과를 얻을 수 있었다.

먼저, 성공한 횟수와 실패한 횟수를 정리한 결과 〔그림 17〕과 같은 모습을 나타냈다. 총 1000번 중 938번을 성공하였으며, 한 번 만에 성공한 경우가 879번, 2번 만에 성공한 경우는 56번, 3번 만에 성공한 경우는 3번으로, 캡차를 올바르게 인식하는데 평균적으로 1.07번 시도하였음을 알 수 있다.

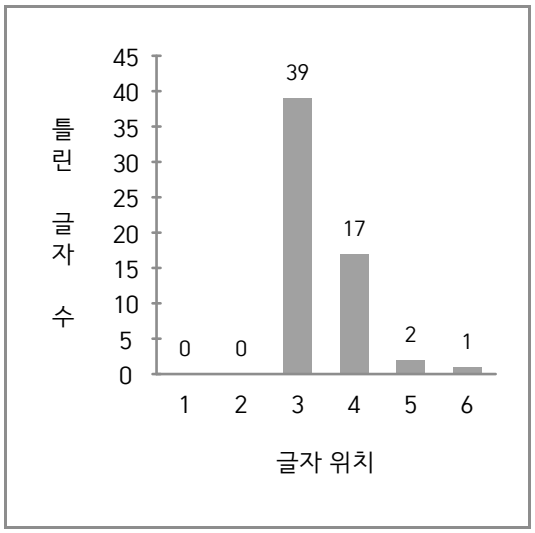


〔그림 17〕 인식하는데 성공하기까지의 시도횟수



〔그림 16〕 틀리게 인식된 글자와 틀린 부분의 원래 글자

다음으로 1000개의 샘플 중 캡차 문자를 잘못 인식한 72개를 대상으로 어떤 글자를 잘못 인식하였는지에 대하여 분석하였고, 데이터를 〔그림 16〕의 도표로 정리하였다. 〔그림 16〕의 도표에서 알 수 있듯이 글자에 따라 어떠한 특성이 있음을 발견할 수 있었다. 도표 내 각 글자의 왼쪽 수치는 '원래 해당 글자였으나 다른 글자로 인식한 경우'이며, 오른쪽 수치는 '원래 다른 글자였으나 해당 글자로 인식한 경우'를 나타내는데, A와 F, N의 경우 해당 글자를 다른 글자로 인식되는 경우가 더 많았고, E, P, V 등의 경우 다른 글자를 해당 글자로 인식하는 경우가 많았음을 알 수 있다. B, D, F, P의 경우 서로 모양이 비슷한 경향을 보여 캡차 이미지 처리 과정에서 글자의 일부가 잘려나가면 다른 글자로 인식될 가능성이 높아지는 것이 그 이유로 보여진다. 또한 V와 N의 경우에 이 글자들이 붙어있으면 어긋나게 잘리는 경우가 늘어나는 경향을 보였다. 이러한 경우 사람이 판단하더라도 정확히 어떤 글자인지 판별하기 쉽지 않은 경우가 다소 있었다.



〔그림 18〕글자 위치에 따른 틀린 글자 수

글자의 위치에 따른 틀린 경우를 분석한 결과, 실험 수행 전 예상한 대로 〔그림 18〕의 도표에서 명확히 알 수 있듯이, 가운데 글자인 3, 4번째 위치한 글자들이 틀리는 경우가 절대 다수였다. 반면 1, 2번째 글자는 전혀 틀리지 않았고, 5, 6번째는 1000회 실험 중 1, 2번 수준으로 극히 적게 틀리는 모습을 보였다.

V. 결 론

이 논문에서는 현재 네이버 카페 가입 과정과 로그인 실패 시 사용하고 있는 캡차를 대상으로 프로그램을 이용한 인식을 시도하여 성공하였다. 즉 프로그램을 이용하여 자동적으로 카페에 가입하고 반복적인 로그인 시도를 컴퓨터를 이용하여 수행할 수 있다는 것을 의미한다. 따라서, 네이버에서 사용하는 캡차가 더 어렵게 바뀌어야 할 것이나, 사용성의 측면에서 얼마나 더 어렵게 바꿀 수 있을지는 연구과제로 남는다. 또한, 이 논문에서 제시한 collapsing을 해결하는 새로운 segmentation 기법을 다른 캡차에 적용하는 것은 향후 연구 주제로 남겨둔다.

참고문헌

〔1〕 Louis von Ahn, Manuel Blum, Nicholas J.Hopper, and John Langford, "CAPTCHA: Using hard AI Problems For Security," Eurocrypt 2003, pp. 294-311

〔2〕 Greg Mori, Jitendra Malik, "Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA," IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 134-141, 2003.

〔3〕 2013 SEO Automation Foundation part 2 Captchas, http://www.campowong.com/2013-seo-automation-foundation-part-2-captchas/

〔4〕 OpenCV, http://opencv.org

〔5〕 LIBSVM, http://www.csie.ntu.edu.tw/~cjlin/libsvm

〔6〕 reCAPTCHA, http://google.com/recaptcha

〔7〕 김성호, 양대헌, 이경희, "색상 정보를 이용한 문자 기반 CAPTCHA의 무력화," 한국정보보호학회 논문지, 19(6), pp. 105-112, 2009년 12월.

〔8〕 John Canny, "A computational approach to edge detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 6, pp. 679-698, 1986.

〔9〕 http://captcha.naver.com/nhncaptcha2.gif

〔10〕 Elie Bursztein, Matthieu Martin, and John C. Mitchell, "Text-based CAPTCHA Strengths and Weaknesses," ACM CCS 2011, pp. 125-138.

〔11〕 J. Yan and A.S. El Ahmad, "A Low-cost Attack on a Microsoft CAPTCHA," ACM CCS 2008, pp. 543 - 554.

〈저자소개〉

양 대 헌 (DaeHun Nyang) 정회원
1994년 2월: 한국과학기술원 과학기술 대학 전기 및 전자 공학과 졸업
1996년 2월: 연세대학교 컴퓨터 과학과 석사
2000년 8월: 연세대학교 컴퓨터 과학과 박사
2000년 9월~2003년 2월: 한국전자통신연구원 정보보호연구본부 선임연구원
2003년 2월~현재: 인하대학교 컴퓨터정보공학과 부교수
〈관심분야〉 암호이론, 암호프로토콜, 인증프로토콜, 무선 인터넷 보안, 네트워크 보안

최 용 헌 (YongHeon Choi) 학생회원
2010년 3월~현재: 인하대학교 컴퓨터정보공학과 학사과정
〈관심분야〉 리눅스, 프로토콜 분석, 디지털 포렌식

홍 석 준 (SeokJun Hong) 학생회원
2011년 3월~현재: 인하대학교 정보통신공학과 학사과정
〈관심분야〉 영상처리, 프로토콜 분석, 디지털 포렌식

이 경 희 (KyungHee Lee) 정회원
1993년 2월: 연세대학교 컴퓨터과학과 학사
1998년 8월: 연세대학교 컴퓨터 과학과 석사
2004년 2월: 연세대학교 컴퓨터 과학과 박사
1993년 1월~1996년 5월: LG소프트(주) 연구원
2000년 12월~2005년 2월: 한국전자통신연구원 선임연구원
2005년 3월~현재: 수원대학교 전기공학과 부교수
〈관심분야〉 바이오인식, 정보보호, 컴퓨터비전, 인공지능, 패턴인식