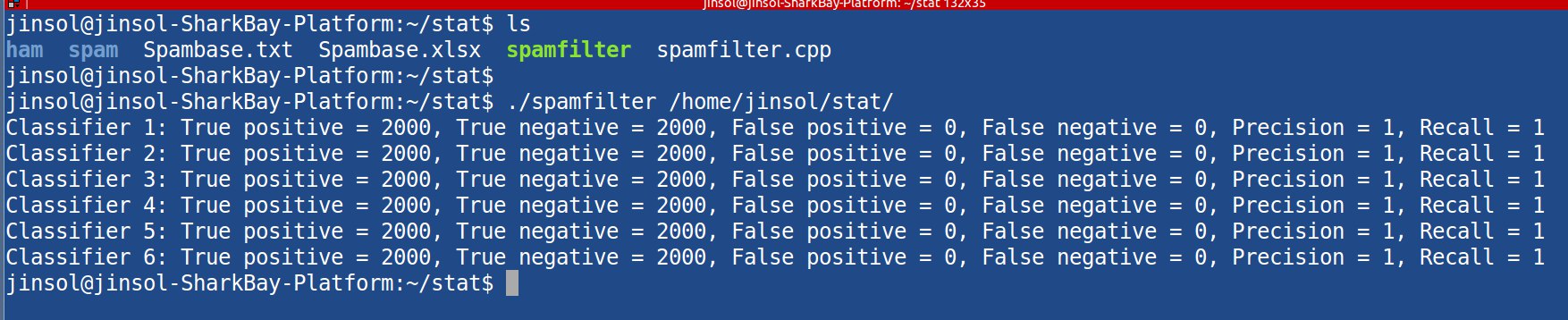
**개발 내용**

**주어진 텍스트가 spam인지 아닌지 판단하는 프로그램**

**필요한 지식**

**Naïve Bayes Theory**

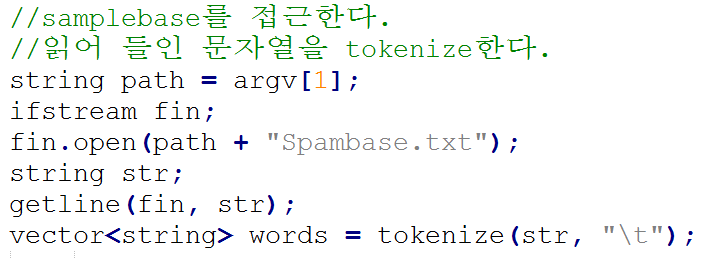
**구현**

****

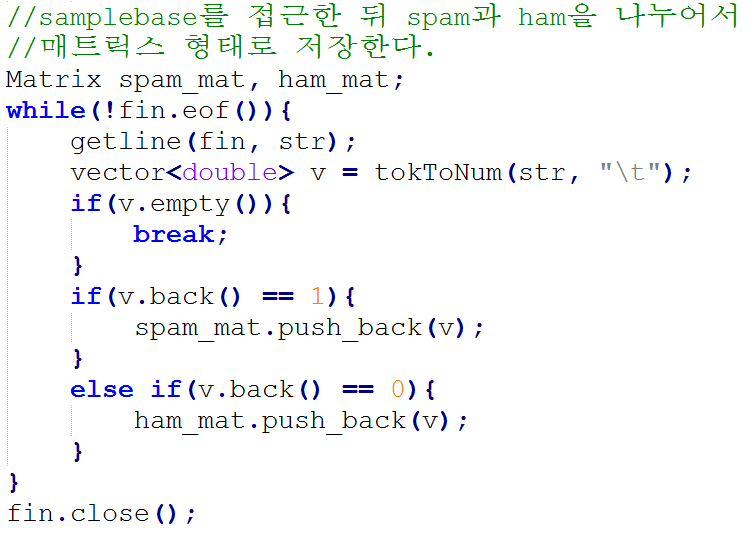
**Stat 폴더 안에 ham, spam, Spambase.txt 그리고 spamfilter.cpp, spamfilter 실행파일이 있을 때**

**Parameter로 file의 경로인 /home/jinsol/stat/을 넣어주면 다음과 같은 결과를 얻을 수 있다.**

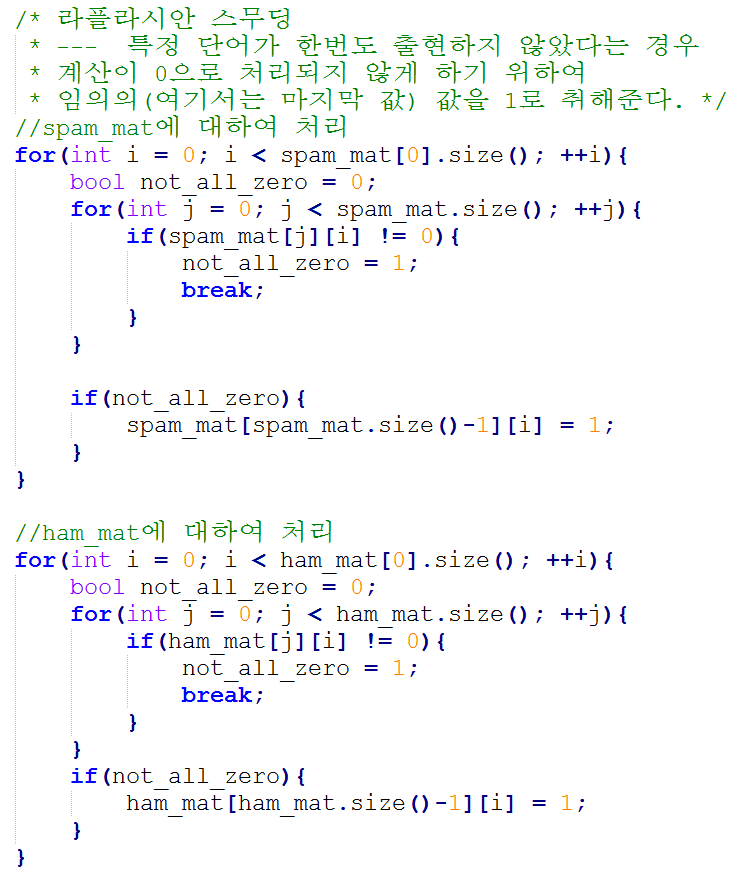
**코드 분석**



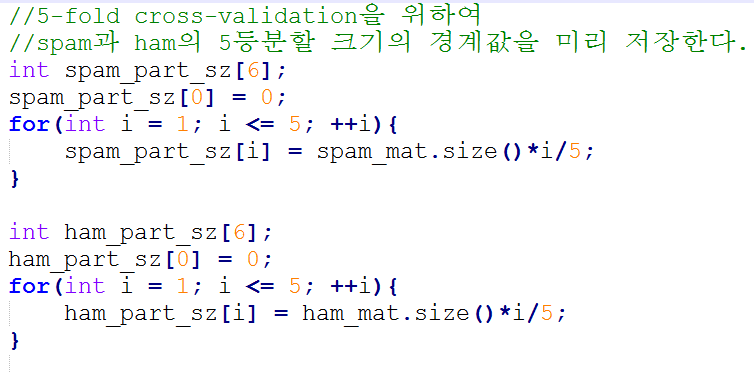
실행파일의 인자로 받은 경로 값을 저장한 뒤 Spambase.txt 파일을 열어서 tokenize한 뒤 문자열 벡터에 담는다.



Samplebase를 통해 입력 받은 데이터를 spam\_mat과 ham\_mat으로 나누어서 담는다. 기준은 is\_spam의 값에 따라 나뉜다.

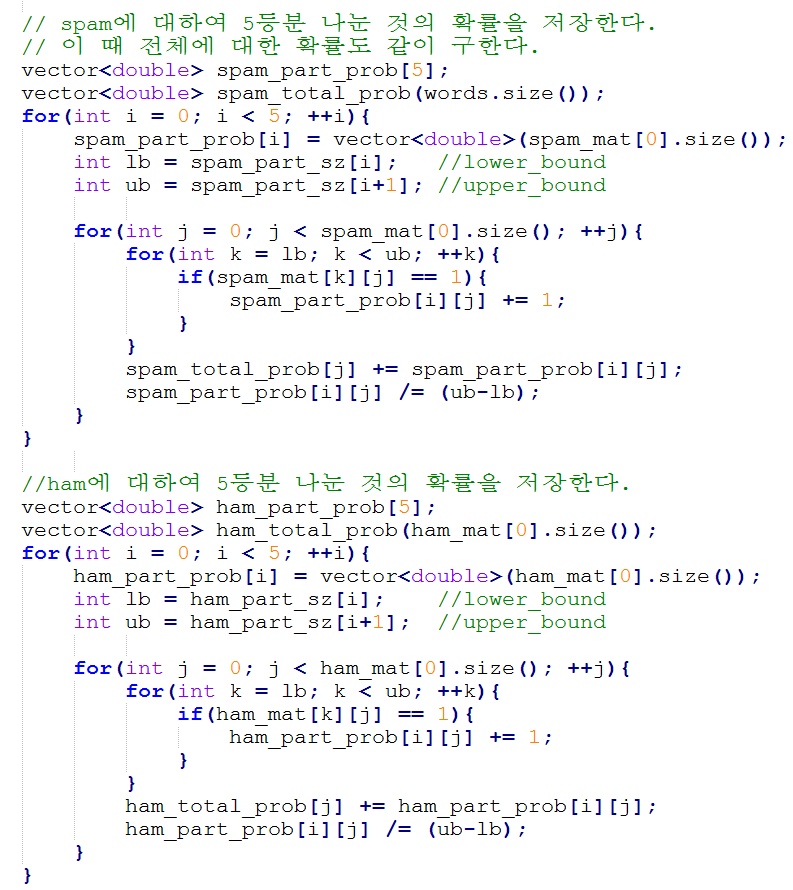


라플라시안 스무딩 기법을 이용한다. 특정 단어가 한번도 출현하지 않은 경우에 나중에 계산할 값이 0이 된다. 0이 되어 버리면 다른 값의 영향이 없어져 버리게 되므로 정밀한 판별을 할 수 없게 된다. 따라서 column의 모든 값이 0이 되는 경우에는 임의의 값 한 개를 1로 만들어서 확률이 0이 안되게 만든다.

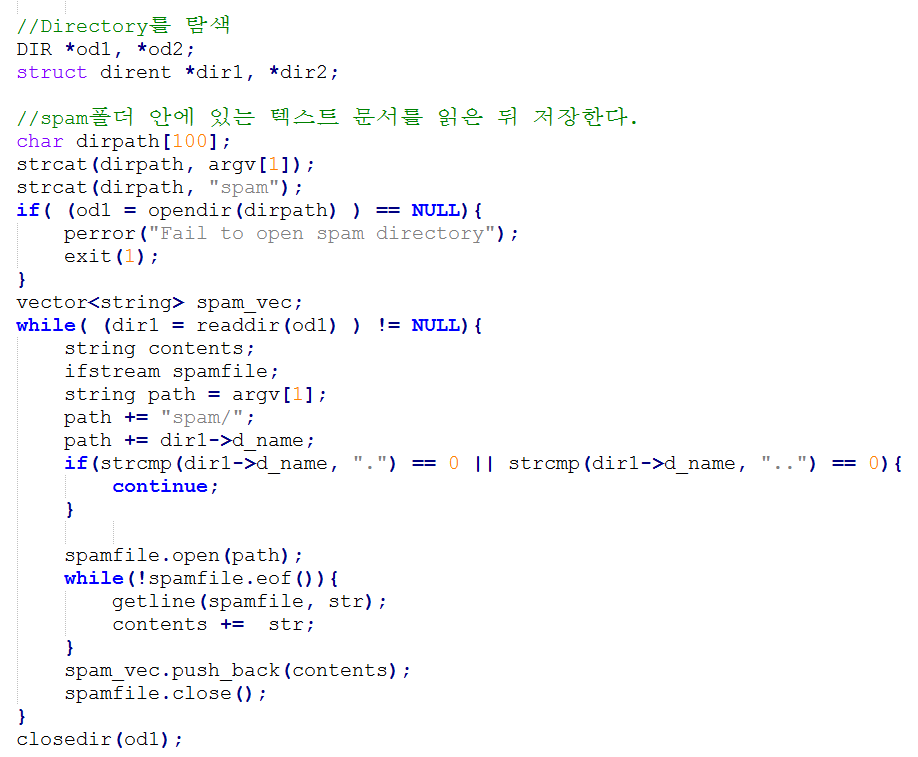


5-fold cross-validation을 수행하기 위하여 spam과 ham을 5등분한다.

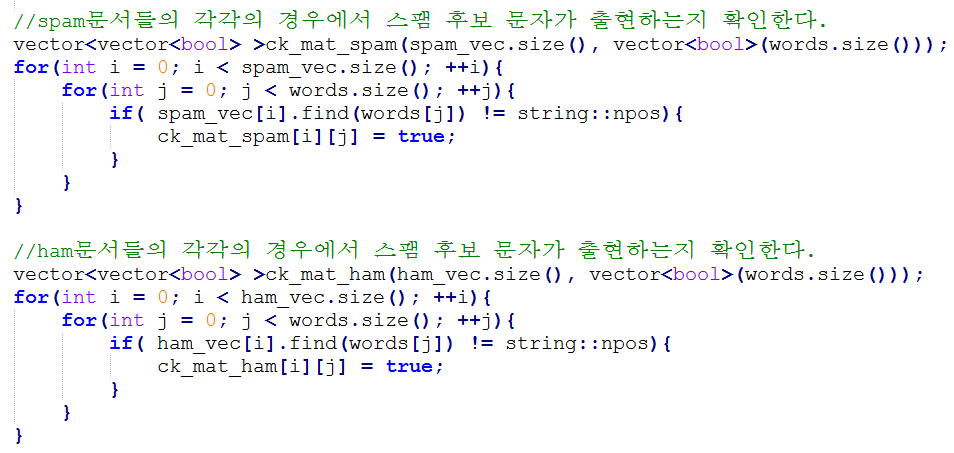
5등분할 경계 값을 미리 저장하여 코드 구현을 용이하게 한다.



Spam과 ham에 대하여 5-fold cross-validation을 수행하기 위하여 5등분에서 각각에 대하여 단어가 나올 확률을 미리 계산해 둔다.

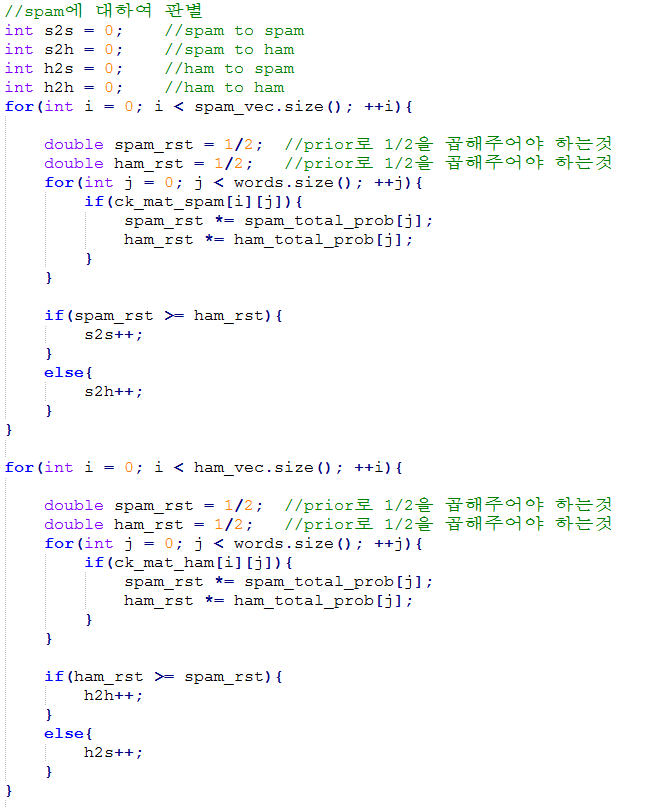


Directory를 탐색하여 spam과 ham 폴더 안에 있는 text파일으르 읽는다. Ham의 경우는 spam과 방법이 동일하므로 코드는 생략하였다.



spam문서들의 각각의 경우에서 spam 후보 문자가 출현하는지 확인한다.

ham 문서들에서도 또한 각각의 경우에서 spam 후보 문자가 출현하는지 확인한다.



위와 같은 방식으로 spam을 판별한다.

예를 들어 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | CS | image | probability |
| 1 | 1 | 0 | 1 |
| 2 | 0 | 1 | 1 |
| 3 | 0 | 1 | 1 |

일 때 각각의 단어의 출현 여부가 [CS = 0, image = 1, probability = 1]인 경우

여기서 1/2 는 prior에 해당한다.

위 과정은 5-fold cross-validation 에서도 같은 방법으로 수행하면 된다.

**개발 환경**

사용언어: C/C++, POSIX

컴파일 환경: g++ 4.8.4 (c++11)

운영체제: **Ubuntu** 14.04

CPU: i5 – 4210 CPU 1.7GHz

RAM: 4.00GB

**결과**

베이지안 이론을 이용하여 스팸필터를 가상으로 구현해보았다. 처음에 라플라시안 스무딩을 하지 않았을 때에는 조금 정확도가 떨어졌는데 라플라시안 스무딩한 뒤 실행해본 결과 정확한 결과를 도출할 수 있었다. 더 나아가서 베이지안 이론을 좀 더 명확하게 이해하여 생략할 수 있는 부분은 없는지 찾아보려고 한다. 또한 컴퓨터공학을 하는 사람의 입장으로서 단지 이론을 구현한 것에 만족하지 않고 더 나은 자료구조로 더 효율적으로 설계할 수 없는지 연구해보겠다.