텍스트 마이닝을 이용한 택배사 스미싱 필터링

김승은, 양지인, 이진영, 최정원

광운대학교 정보융합학부

ABSTRACT. 스미싱이란 문자메시지를 이용한 피싱을 의미한다. 다양한 스미싱 문자 유형 중 택배사를 사칭한 스미싱은 방송통신위원회 통계로 2016 년에서 2018 년 8 월, 약 1 년 반 동안 75%를 차지할 만큼 많은 비율을 차지하고 있다. 본 보고서에서는 택배사 사칭 스미싱 필터링에 어떤 알고리즘이 가장 정확한 결과를 낼 수 있는지 알아보기 위해 알고리즘을 R 언어로 구현하고 그 정확도에 대해 알아보고자 한다. 이를 바탕으로 택배사 사칭 스미싱 문자 데이터를 수집하여 4 개의 알고리즘을 적용하고, 어떤 알고리즘이 택배사 사칭 스미싱 필터링에 가장 적합한지 제안한다.

Keywords: 텍스트마이닝, 택배, 스미싱, SVM, Knn, Naïve Bayes,

1. 서론

스미싱(SmiShing)이란 SMS(문자메시지)와 Phishing 의 합성어로, 문자메시지를 이용한 피싱을 의미한다[1]. 주로 사용자의 확인을 유도하는 내용과 단축 URL 을 함께 보내 URL 을 확인 시 개인정보를 탈취해가거나 소액결제를 악용해 피해를 입히는 신종 사기수법이다.

모바일 청첩장, 민원 신고 내역 확인, 유출 정보 확인 등 스미싱 문자에도 많은 유형이 있지만, 방송통신위원회에 따르면 2016 년부터 2018 년 8 월까지 탐지된 스미싱 문자 97 만 5050 건 중 택배 배송과 관련한 것이 약 72 만 건을 차지한다.[2] 또한 이스트 시큐리티 공식 블로그의 주간 스미싱 통계를 보았을 때, 택배사 사칭 문자가 현재까지도 1 위를 차지하는 만큼 택배사 스미싱은 여전히 기승을 부리고 있다.[3]
택배사를 사칭한 스미싱은 아래 [그림 1]과 같은 내용으로 수신되고 있다.

[그림 1. 택배사 사칭 스미싱 예시 및 링크 접속 시 홈페이지]

미배달, 주소지 오류만이 아닌 배송불가, 주소지 확인만이 아닌 택배 유실, 압류 등 다양한 내용을 통해 사용자들을 스미싱에 넘어가도록 유도하고있다.

스미싱 사기에 대한 절차는 다음과 같다. URL 이 포함된 문자 메시지가 수신되었을 경우, 해당 URL 을 클릭하면 해커가 만든 서버에 접속하여 스마트 폰에 악성코드를 다운로드한 후 설치를 유도한다. 그 후 해커는 해당 앱을 통해 이용자의 정보를 탈취하고, 소액결제나 번호도용 등과 같은 곳에 사용한다.

앞선 언급한듯이, 스미싱 문자 중 택배 스미싱이 70% 이상을 차지할 정도로 많은 양이며, 주변에서 택배사기 문자를 받거나 실제로 URL 을 눌러 피해를 입은 사례가 많기 때문에 택배사를 사칭한 스미싱 문자 필터링의 필요성을 느꼈다. 따라서 본 텍스트마이닝 프로젝트에서는 택배 스미싱 문자 데이터와 일반 택배 문자 데이터를 수집하여 전처리한 뒤, 다양한 텍스트마이닝 알고리즘에 적용하고 그 중 가장 효율이 높은 알고리즘을 찾는 것을 목표로 한다.

2. 데이터 수집 방법 및 데이터 구성요소

2.1. 데이터 수집 과정

택배사 사칭 스미싱은 대부분 문자 내역을 캡쳐한 이미지 파일로 인터넷 상 업로드 되어있다. 구글, 네이버, 다음, SNS 에서 검색된 스미싱 캡쳐 이미지를 기반으로 텍스트를 직접 타이핑해서 택배사 사칭 문자 153 개, 정상 문자 80 개를 추출해냈다. 따라서 총 233 개의 문서를 얻어냈다. 이제 이 문서에서 택배사 사칭 스미싱 데이터를 줄여 spam 이라고 부르고, 일반 택배사 데이터는 ham 이라고 지칭하겠다.

[Web발신] [로젠택배] 등기 소포 수취 불가 상태입니다. 주소지 확인/변경 요망 http://me2.do/xR3ss517
[Web발신] [CJ대한통운] 홍길동 고객님 택배 배송 불가 (도로명 불일치) 주소지 확인))) https://goo.gl/4YfrWP 11:40
고객님 택배가 반송 처리 되었으니 주소를 수정 하세요 homehanex.blogspot.kr
추석 물량 증가로 배송이 지연되고 있습니다. 배송 일정 확인 하세요 http://goo.gl/b1j1xi
고객님 택배가 도착하였습니다. 배송조회☞ www.kevv.pw
[CJ대한통운] 운송장 번호 [690*27] 주소지 미확인..반송 처리 주소 확인. cjjfgs.wksla.com
[Web발신] [한진택배] (도로명 불일치) 배송 불가 변경 요망 http://goo.gl/Tb5wo3
[한진택배] 등기 우편 수취 불가(주소지 불명). 주소지 변경/확인 바랍니다.http://me2.do/GPtfB2dd
[대한통운 택배] 미수령 택배가 있습니다 확인 해 주시길 바랍니다. http://ko*e*to*ls.c*c*r/*p*oa*/p*y/

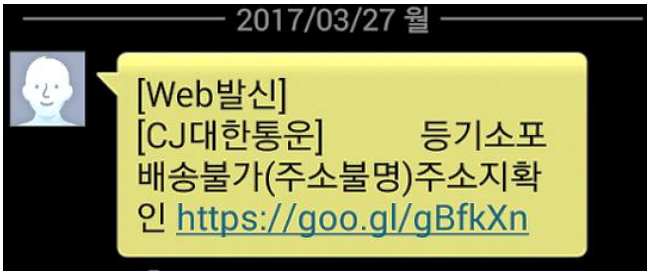
[그림 2.1. 택배사 사칭 스미싱 데이터 예시]

집앞 슈퍼에서 택배 찾아 가세여
[Web발신] 편의점 택배 어제 수거 완료
고객님 택배를 23784927를 오늘 배달 예정 입니다. 밀양 우체국
죄송합니다. 혹한 물량 증가로 배송이 지연 중으로 1~2일 내 배송 예정 입니다
옐로우캡택배 84183740로 발송 로제컴퍼니 관세청 통관 후 이동시 조회 되세요
[Web발신] 기쁜 소식 알려 드립니다! KGB택배 4385로 발송 되었습니다 mixxmix
택배 경비실에 맡겨 두었습니다 12324 현대택배

[그림 2.2. 택배 정상문자 데이터 예시]

2.2. 데이터 구성 요소

가져온 spam 데이터는 크게 [발신 기기 종류] / 문자내용 / 링크로 나누어 볼 수 있다.



[그림 3. 택배사 사칭 스미싱 예시]

3.연구방법

3.1. 데이터 전처리 방법 및 과정

본 프로젝트에서는 분석을 위한 툴로 R 을 사용하였다. 의미 있는 단어만을 추출하는 것을 목적으로 데이터 전처리를 진행하였다. 우선 모든 문서가 들어있는 “Dingdong.txt” 파일을 불러온다. 그 후 R 의 내장함수를 사용하여 중복되는 문서와 문서 내의 특수문자를 제거하였다.

```
> ##파일 불러오기
> ding <- readLines("Dingdong.txt")
> head(ding)
[1] "[Web발신] [로젠택배] 등기 소포 수취 불가 상태입니다. 주소지 확인/변경 요망 http://me2.do/xR3ss517"
[2] "[Web발신] [CJ대한통운] 홍길동 고객님 택배 배송 불가 (도로명 불일치) 주소지 확인 >>> https://goo.gl/4YfrWP 11:40"
[3] "고객님 택배가 반송 처리 되었으니 주소를 수정 하세요 homehanex.blogspot.kr"
[4] "추석 물량 증가로 배송이 지연되고 있습니다. 배송 일정 확인 하세요 http://goo.gl/b1j1xi"
[5] "고객님 택배가 도착하였습니다. 배송조회☞ www.kevv.pw"
[6] "[CJ대한통운] 운송장 번호 [690*27] 주소지 미확인..반송 처리 주소 확인. cjjfgs.wksla.com"
> ##중복제거, 특수기호 제거
> ding2 <- unique(ding)
> ding2 <- str_replace_all(ding,"[^[:alpha:][:blank:]]"," ")
> head(ding2)
[1] " Web발신   로젠택배  등기 소포 수취 불가 상태입니다  주소지 확인 변경 요망 http   me  do xR ss   "
[2] " Web발신   CJ대한통운  홍길동 고객님 택배 배송 불가  도로명 불일치  주소지 확인    https   goo gl  YfrWP     "
[3] "고객님 택배가 반송 처리 되었으니 주소를 수정 하세요 homehanex blogspot kr"
[4] "추석 물량 증가로 배송이 지연되고 있습니다  배송 일정 확인 하세요 http   goo gl b j xi"
[5] "고객님 택배가 도착하였습니다  배송조회  www kevv pw"
[6] " CJ대한통운  운송장 번호          주소지 미확인  반송 처리 주소 확인  cjjfgs wksla com"
```

이후 R 의 패키지인 “KoNLP”를 사용해 명사 단위로 분리한 뒤, 문서 내의 영어를 제거했다.

```
> ##단어로 분리후 영어제거
> ding2 <- extractNoun(ding2)
> ding2 <-lapply(ding2,function(x){gsub("[^가-힣, ]"," ",x)})
> head(ding2)
[[1]]
 [1] "   발신"  "로젠택배" "등기"     "소포"     "수취"     "불"       "상태"     "주소"     "확인"     "변경"     "요망"
[12] "    "    "  "    "  "    "  "    "  "    "  "    "  "    "  "

[[2]]
 [1] "   발신" "  "     "대한"    "통운"    "홍길동"  "고객"    "님"      "택배"    "배송"    "불"      "도로명"  "불일치"  "주소"
[14] "확인"    "    "   "  "    "  "    "  "    "  "    "  "

[[3]]
 [1] "고객"     "님"       "택배"     "반송"     "처리"     "주소"     "수정"     "          "  "  "        "  "  "
[11] "  "
```

명사로 분리 시 한 글자로만 분리되거나, 너무 길게 분리된 것을 제거했다.

```
> ##한 글자, 긴 문자 제거
> ding2 <- lapply(ding2, function(x){
+   Filter(function(y){nchar(y)<=10&nchar(y)>1},x)
+ })
> head(ding2)
[[1]]
 [1] "   발신"  "로젠택배" "등기"     "소포"     "수취"     "상태"     "주소"     "확인"     "변경"     "요망"     "    "
[12] "  "       "  "       "  "

[[2]]
 [1] "   발신" "  "      "대한"    "통운"    "홍길동"  "고객"    "택배"    "배송"    "도로명"  "불일치"  "주소"    "확인"    "     "
[14] "    "     "  "      "   "

[[3]]
[1] "고객"     "택배"     "반송"     "처리"     "주소"     "수정"     "         " "  "       "
```

2 차원 리스트에서 차원을 삭제하여 1 차원 리스트로 변환한 후 콤마를 제거했다.

```
> ##정렬하지 않고 이중리스트를 1차원의 리스트로 변환(tm에 넘겨주기 위한 작업) 후 콤마(,) 제거
> ding2 <- format(ding2,justify = "none")
> ding2 <- gsub("\\,","",ding2)
> head(ding2)
[1] "   발신 로젠택배 등기 소포 수취 상태 주소 확인 변경 요망                "
[2] "   발신   대한 통운 홍길동 고객 택배 배송 도로명 불일치 주소 확인                    "
[3] "고객 택배 반송 처리 주소 수정                          "
[4] "추석 물량 증가 배송 지연 배송 일정 확인              "
[5] "고객 택배 도착 배송 조회          "
[6] "   대한 통운 운송장 번호 주소 미확인 반송 처리 주소 확인                  "
```

이후 “tm”패키지의 내장함수인 corpus 를 이용하여 문서 집합을 생성한 후, document – term matrix 형태로 만들었다. 이 때 Knn, SVM, K-Means 는 weighting 을 TF-IDF 로 설정하고, Naïve Bayes 는 Binary 로 설정하였다.

```
##tm을 통해 코퍼스로 만든다.
ding3 <- Corpus(VectorSource(ding2))

##Doc-Term Matrix 만들기 (weighting을 kmeans,knn,svm은 tf-idf, naive bayes는 바이너리)
ding_tfidf <- DocumentTermMatrix(ding3, control = list(weighting = function(x)
  weightTfIdf(x, normalize = F)))
ding_bin <- DocumentTermMatrix(ding3,control = list(weighting=function(x)
  weightBin(x)))
View(ding_tfidf)
View(ding_bin)
```

Document-Term Matrix 생성을 통해 생긴 단어들 중 출현 빈도가 적은 단어들을 삭제했다. 그 결과 86 개의 단어들로 추려졌고, 이 단어들을 기반으로 분석을 진행할 것이다.

```
##Term sparse(단어 출현 확률이 99%에 들지 않으면 제거) ##단어 86개
library(class)
cleandtm = removeSparseTerms(ding_tfidf,0.99)
cleandtm2 = removeSparseTerms(ding_bin,0.99)
cleandtm2$dimnames
```

```
$Terms
 [1] "등기"          "로젠택배"      "발신"          "변경"          "상태"          "소포"          "요망"
 [8] "주소"          "확인"          "고객"          "대한"          "도로명"        "배송"          "불일치"
[15] "택배"          "통운"          "반송"          "수정"          "처리"          "물량"          "증가"
[22] "지연"          "추석"          "도착"          "조회"          "미확인"        "번호"          "운송장"
[29] "한진택배"      "불명"          "미수"          "주시"          "바람"          "재확인"        "회송"
[36] "부재중"        "정보"          "물품"          "예정"          "오후"          "미배달"        "주문"
[43] "발송"          "드립니"        "보상"          "분실"          "안내"          "모호"          "본인"
[50] "압류"          "문자"          "배달"          "우체국"        "상품"          "우체"          "물건"
[57] "수령"          "정확"          "잘못"          "안녕"          "이름"          "전달"          "두었습니"
[64] "현관"          "보관"          "소화"          "오늘"          "죄송"          "경비실"        "현대"
[71] "방문"          "예정입니"      "경비"          "금일"          "연락"          "하루"          "죄송합니"
[78] "부재"          "부탁"          "장소"          "소중"          "감사합니"      "전화"          "한진택배입니다"
[85] "바랍니"        "함동"
```

[그림 4. 추출된 단어 86 개]

이후 두 데이터를 Dataframe 형식으로 변환했다.

```
> ##데이터프레임으로 변환
> cleanframe <- as.data.frame(as.matrix(cleandtm))
> cleanframe2 <- as.data.frame(as.matrix(cleandtm2))
>
```

	등기	로젠택배	발신	변경	상태	소포	수취	요망	주소	확인	고객	대한	도로명	배송	불일치	택배	통운	홍길동	반송
1	4.398549	3.688056	1.927244	3.334419	6.273018	4.536053	6.857981	3.95109	1.366128	1.465664	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	1.927244	0.000000	0.000000	0.000000	0.000000	0.00000	1.366128	1.465664	1.728698	1.214125	3.688056	1.903785	3.857981	0.8807011	1.130061	7.857981	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	1.366128	0.000000	1.728698	0.000000	0.000000	0.000000	0.000000	0.8807011	0.000000	0.000000	2.813587
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	1.465664	0.000000	0.000000	0.000000	3.807569	0.000000	0.0000000	0.000000	0.000000	0.000000
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	1.728698	0.000000	0.000000	1.903785	0.000000	0.8807011	0.000000	0.000000	0.000000
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	2.732256	1.465664	0.000000	1.214125	0.000000	0.000000	0.000000	0.0000000	1.130061	0.000000	2.813587
7	0.000000	0.000000	1.927244	3.334419	0.000000	0.000000	0.000000	3.95109	0.000000	0.000000	0.000000	0.000000	3.688056	1.903785	3.857981	0.0000000	0.000000	0.000000	0.000000
8	4.398549	0.000000	0.000000	3.334419	0.000000	0.000000	6.857981	0.00000	2.732256	1.465664	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000000	0.000000	0.000000	0.000000
9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	1.465664	0.000000	1.214125	0.000000	0.000000	0.000000	1.7614021	1.130061	0.000000	0.000000
10	4.398549	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	1.366128	1.465664	0.000000	1.214125	0.000000	0.000000	0.000000	0.8807011	1.130061	0.000000	0.000000
11	0.000000	3.688056	1.927244	0.000000	0.000000	0.000000	0.000000	0.00000	2.732256	1.465664	1.728698	0.000000	0.000000	1.903785	0.000000	0.8807011	0.000000	0.000000	0.000000

[그림 5. Knn, SVM, K-Means 용 TF-IDF 데이터 cleanframe]

	등기	로젠택배	발신	변경	상태	소포	요망	주소	확인	고객	대한	도로명	배송	불일치	택배	통운	반송	수정	처리	물량	증가	지연	추석	도착	조회	미확인	번호	운송장	한진택배	불명	미수	주시	바람	재확인
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
7	0	0	1	1	0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
8	1	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
10	1	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
11	0	1	1	0	0	0	0	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
12	1	0	1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
13	1	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

[그림 5.1. Naïve Bayes 용 Binary 데이터 cleanframe2]

각 document 에 따른 라벨링을 하기 위해 Class 라는 열을 추가하여 ham, spam 을 넣어주었다. 문서 집합에 153 개의 ham 데이터와 80 개의 spam 데이터를 순서대로 넣었으므로 같은 순서대로 라벨을 붙여주었다.

```
> ##txt파일에 순서대로 153개의 spam과 80개의 ham을 넣었으므로 그대로 라벨 지정
> cleandata <- cbind("Class" = c(rep("spam",153),rep("ham",80)), cleanframe)
> cleandata2 <- cbind("Class" = c(rep("spam",153),rep("ham",80)), cleanframe2)
>
```

cleandata

Filter

	Class	등기	로젠택배	발신	변경	상태	소포	요망	주소	확인	고객	대한	도로명	배송	불일치
1	spam	4.404755	3.694261	1.933449	3.340624	6.279224	4.542258	3.957296	1.372333	1.454795	0.000000	0.000000	0.000000	0.00000	
2	spam	0.000000	0.000000	1.933449	0.000000	0.000000	0.000000	0.000000	1.372333	1.454795	1.734903	1.205975	3.694261	1.90999	
3	spam	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.372333	0.000000	1.734903	0.000000	0.000000	0.00000	
4	spam	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.454795	0.000000	0.000000	0.000000	3.81998	
5	spam	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.734903	0.000000	0.000000	1.90999	
6	spam	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.744666	1.454795	0.000000	1.205975	0.000000	0.00000	
7	spam	0.000000	0.000000	1.933449	3.340624	0.000000	0.000000	3.957296	0.000000	0.000000	0.000000	0.000000	3.694261	1.90999	
8	spam	4.404755	0.000000	0.000000	3.340624	0.000000	0.000000	0.000000	2.744666	1.454795	0.000000	0.000000	0.000000	0.00000	
9	spam	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.454795	0.000000	1.205975	0.000000	0.00000	

[그림 6. 라벨링 된 Knn, SVM 용 TF-IDF 데이터 cleandata]

cleandata2

Filter

	Class	등기	로젠택배	발신	변경	상태	소포	요망	주소	확인	고객	대한	도로명	배송	불일치	택배	통운	반송
1	spam	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	
2	spam	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1	1	
3	spam	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	
4	spam	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	
5	spam	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	
6	spam	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1	
7	spam	0	0	1	1	0	0	1	0	0	0	0	1	1	1	0	0	
8	spam	1	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	
9	spam	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1	

[그림 7. 라벨링 된 Naïve Bayes 용 Binary 데이터 cleandata2]

비지도 학습인 K-Means 의 경우 라벨링이 필요하지 않으므로 라벨링 하지 않은 데이터를 사용한다.

지도 학습 데이터를 위해 전체 문서의 60%를 트레이닝, 40%를 테스트 데이터로 나누었다. 선택 방식은 random 이다. 결과에 나타난 숫자는 train data 와 test data 에 속한 문서의 번호를 의미한다.

```
#전체 데이터 중 랜덤으로 60%는 트레이닝, 40%는 테스트 (뽑은 샘플은 document의 번호를 리턴한다.)
train <- sample(nrow(cleandata),ceiling(nrow(cleandata)*0.60))
test = (1:nrow(cleandata))[-train]
```

```
> train
  [1] 226  98 231   8  27  45 132 171 170  96  12  26 153 156  67  58  20 220  52 183  68  44 101  94 164 228 219  29 138  30
 [31] 107  39 155 139  34 160 206   4  90 191 169 181 114  74 127 190 150 115 133  62 154   6 144 222  40 146  33  85  21 118
 [61]  43  36 151 111 149 189 106   9  69  66 142  83 208 229  80 112 174 165 162 224  35 110  11  78 152  57 126 233 210 123
 [91] 203 143  56   1 102  82  73 188  10  61 192 178 109  49 207  89 168  55 185 198 227 119 213 204 113 104  65  28  46  53
[121]  50  48 177  25  63 202 159  22   2  91  87  76 130  93 167 176  38 193   3 184
> test
  [1]   5   7  13  14  15  16  17  18  19  23  24  31  32  37  41  42  47  51  54  59  60  64  70  71  72  75  77  79  81  84
 [31]  86  88  92  95  97  99 100 103 105 108 116 117 120 121 122 124 125 128 129 131 134 135 136 137 140 141 145 147 148 157
 [61] 158 161 163 166 172 173 175 179 180 182 186 187 194 195 196 197 199 200 201 205 209 211 212 214 215 216 217 218 221 223
 [91] 225 230 232
```

3.2. 알고리즘 선택 배경 및 구현 내용

현재 휴대폰 내 스미싱 메시지 필터링 방식은 특정 문자열이 문자 메시지에 있을 경우에 스팸으로 분류하거나, 특정 번호를 입력하여 스팸 문자 여부를 판단하는 방식으로 지원을 하고 있다. 하지만 이러한 방식만으로는 현재 발송되고 있는 스팸 문자 메시지에 제대로 된 대처를 해주기 힘든 실정이다.

그래서 우리는 수업시간에 배웠던 비지도 학습 k-means 와 지도학습 Knn, Naive bayes, SVM 을 사용하며 각 알고리즘이 지닌 특성과 방식을 이용하여 결과를 예측해보고 확인하려고 한다.

3.3.분석 과정

3.3.1. 분석 프로그램 선택 및 분석 과정 설명

분석 코드는 파란 글씨, 주석과 설명은 검은 글씨로 나타난다.

3.3.1.1. K-Means

k-means 기법은 비지도학습의 기법이므로, 데이터셋이 spam 혹은 ham 인지에 대한 정보를 주지 않고 시작한다. document-document matrix 를 만들어서 각 문서 간의 거리가 얼마인지 코사인 유사도를 통하여 구한다. 그리고 k=2 로 설정하여 두 개의 군집으로 나뉜 결과를 보고, 실제 데이터셋과 비교하였을 때 이 기법을 통해 잘 군집화 되었는지에 대한 결과를 확인한다.

#앞서 시행했던 전처리에서 cleanframe 이라는 변수를 갖고 온다. 그 행렬에서 문서 간의 거리를 나타낼 수 있는 document-document 행렬을 만들고자 dist 라는 함수를 써서 cosine distance 의 값을 갖는 행렬을 만들어 coco 에 저장한다. 이때 cosine 의 계산 방식은, '1 - (코사인 유사도)' 이다. (dist 라는 함수를 쓰기 위해서는 "proxy"라는 패키지를 불러온다.)

```
library(proxy)
coco = as.matrix(dist(cleanframe, method = "cosine"))
View(coco)
```

##위에서 구한 coco 를 기반으로 2 개의 군집을 만들도록 kmeans 를 실행한다. (우리가 분류하고자 하는 속성이 ham 과 spam, 2 개이므로 k=2)

```
kmeans_test <- kmeans(coco, 2)
```

#kmeans 의 실행결과를 그래프를 통해 시각적으로 나타낸다.

```
library(cluster)
clusplot(coco,kmeans_test$cluster,color=TRUE,main="K-means clustering
(k=2)",cex=2,shade = TRUE)
```

#실제 데이터와 얼마나 차이가 있게 분류되었는지 테이블로 확인한다.

```
table(cleandata$Class,kmeans_test$cluster)
```

#어떤 k 값을 가져야 잘 분리가 될 수 있는지에 대한 반복문이다. k 의 최댓값은 50 으로 설정하고, kmeans 를 실행하였을 때 생성된 각 군집의 중심과 각 군집에 속한 개체간 거리의 제곱합을 sumwit 에 저장한다. 50 번을 반복해 나온 값들을 그래프로 그린 후, 그래프가 완만해지는 elbow 를 찾는다.

```
k_max<-50
sumwit <- sapply(1:k_max,function(k){kk<-kmeans(coco,k,nstart =
10,iter.max = 9999)$tot.withinss})
plot(1:k_max,sumwit,xlab="Number of K", ylab="Total within-clusters sum of
squares",
     main="Elbow method to find K", type="o",pch=20)
text(1:k_max,sumwit,pos=3)
```

3.3.1.2. Knn(k Nearest Neighbor)

Knn 기법은 k-nearest neighbor 의 약자로 k 의 숫자를 사람이 임의로 정한 뒤, 분류하려는 데이터를 기준으로 가까운 k 개의 데이터를 기반으로 해서 분류하는 알고리즘이다. 이러한 기법을 R 에서 “class”라는 패키지 안에 내장되어 있는 Knn 함수를 통해 구현할 수 있다. 전처리에서 설정한 트레이닝 데이터와 테스트 데이터를 기반으로 k 의 값에 따라 이 기법이 테스트 데이터를 얼마나 잘 분류할 수 있는지에 대한 정확도가 달라지므로, 반복문을 통해 가장 높은 정확도를 도출해 낼 수 있는 k 의 값을 찾는다. 그리고 그 때의 정확도를 구해서 성능평가를 한다.

#Knn 분석에 필요한 클래스인 “class”패키지를 불러온다.

```
library(class)
```

#트레이닝 데이터와 테스트 데이터를 분리한다. 변수명에 ‘label’이 들어가 있는 것은 ham 인지 spam 인지에 대한 Class 값이 들어가 있다.

```
train_data = cleandata[train,2:87]
test_data = cleandata[test,2:87]
train_label = cleandata[train,1]
test_label = cleandata[test,1]
```

#정확도가 가장 높을 때의 k 값을 구하는 코드이다. 이때 주의할 점은, 전처리에서 샘플을 새로 설정할 때마다 적정한 k 값이 달라진다는 것이다.
#트레이닝 데이터의 개수만큼 반복문을 만든다. Knn_k 에는 knn 알고리즘의 결과를 넣는다. 그러고 knn 을 통해 분류한 라벨이 실제 테스트 라벨처럼 잘 분류가 되었는지에 대한 정확도 확률을 acc_k 에 넣는다. 이러한 k 의 값과 acc_k 의 값을 dataframe 으로 만들어 그래프로 그려내면, k 가 어떤 값을 가질 때 가장 높은 정확도를 갖는지 시각적으로 알 수 있다.

```
acc_k <- NULL
for(kk in c(1:nrow(train_data))){
  set.seed(1234)
  Knn_k <- Knn(train_data, test_data, train_label, k = kk)
  acc_k <- c(acc_k, sum(Knn_k == test_label) / length(test_label)) }
valid_k <- data.frame(k = c(1:nrow(train_data)), accuracy = acc_k)
plot(formula = accuracy ~ k, data = valid_k, type = "o", pch = 20, main =
"k 값에 따른 정확도")
with(valid_k, text(accuracy ~ k, labels = rownames(valid_k), pos = 3, cex =
0.7))
```

#위에서 나온 결과에 따라 정확도가 가장 높을 때의 k 를 사용해 Knn 테스트를 시행한다. 이때 당시에는 k 가 11 일 때, 가장 높은 정확도를 갖는다고 나왔다.

```
Knntest = Knn(train_data, test_data, train_label,k=11)
```

#Knntest 에 저장된 값들을 실제 테스트 데이터 셋과 비교하여 볼 수 있는 테이블을 그린다.

```
library(gmodels)
CrossTable(test_label,Knntest,prop.chisq = F)
```

3.3.1.3. Naïve Bayes

Naïve Bayes R 에서 베이지언 분류 모델을 지원해주는 e1071, STAN 같은 다양한 패키지가 존재한다. 그 중에서 가장 자주 사용되는 David Meyer 의 e1071 패키지에 내장된 NaiveBayes 함수를 사용하여 정상 문자와 비정상 문자를 구분한다. 수집한 데이터가 적어 어휘의 수 역시 적을 것으로 판단하여 베르누이 나이브 베이즈 방법을

이용해 분류기를 생성한다. 먼저 단어의 존재 여부를 나타내는 DTM(Document-Term Matrix)를 이진 데이터로 나타낸다. 또한 나이브 베이즈 분류기는 일반적으로 범주형 특징으로 된 데이터에 대해 훈련을 하기 때문에 이진 데이터로 나타낸 DTM 을 단어의 존재 여부에 따라 "Yes"혹은"No"로 나타내는 범주형 데이터로 변환하는 과정을 거친다. 범주형으로 나타낸 최종 DTM 데이터를 전처리 과정에서 정한 트레이닝 데이터와 테스트 데이터를 기반으로 나누어 추출한 후, 라플라스 상수의 사용 여부에 따라 2 개의 나이브 베이즈 분류기로 생성한다. 마지막으로 테스트 데이터를 이용하여 2 개의 분류기로 예측을 한 후 성능평가를 통해 각 모델을 비교한다.

#나이브 베이즈 분류기는 일반적으로 범주형 특징으로 된 데이터에 대해 훈련한다. 따라서 단어의 존재 여부에 따라 "Yes" 혹은 "No"를 나타내는 범주형 변수로 바꿀 필요가 있으므로 셀의 값이 0 보다 클 경우(=1 일 경우) "Yes", 그 외의 값(=0 일 경우)은 "No"로 나타낸다.

```
convert_counts <- function(x) { x <- ifelse(x > 0, "Yes", "No") }
```

#이진으로 되어있는 데이터를 위에서 정의한 convert_counts 를 적용하여 범주형으로 나타낸다. 행이나 열을 명시하기 위해 MARGIN 사용했다. 열에 관심 있으므로 MARGIN 을 2 로 지정했다.

```
sms_train <- apply(naive_train_data, MARGIN = 2, convert_counts)
sms_test <- apply(naive_test_data, MARGIN = 2, convert_counts)
```

#라플라스 상수 유무에 따른 2 개의 분류기를 생성한다.

```
library(e1071)
sms_classifier <- naiveBayes(sms_train, as.factor(naive_train_label))
sms_classifier2 <- naiveBayes(sms_train, as.factor(naive_train_label),
laplace=1)
```

#모델 성능 평가를 하기 위해 각 분류기로 예측을 생성한 후, 예측된 값과 실제 값을 비교한다. 예측을 위해 predict 함수를 사용한다.

```
sms_test_pred <- predict(sms_classifier, sms_test)
sms_test_pred2 <- predict(sms_classifier2, sms_test)
```

#예측을 실제값과 비교하기 위해 gmodels 패키지의 CrossTable() 함수를 사용한다.

```
library(gmodels)
CrossTable(sms_test_pred, naive_test_label,
           dnn = c('predicted', 'actual'))
CrossTable(sms_test_pred2, naive_test_label,
           dnn = c('predicted', 'actual'))
```

3.3.1.4. SVM(Support Vector Machine)

SVM은 학습 데이터를 두 개의 집단으로 분리시키는 최적의 초평면을 찾는 이진 분류기로서 우수한 성능 때문에 다양한 분야에서 귀납 추론, 이진 분류, 예측 등을 목적으로 사용되는 알고리즘이다. 또한 대표적인 블랙박스 모델 중 하나이기 때문에 학습 후 생성되는 SVM의 해석에 대한 연구도 활발히 진행되고 있다. SVM 알고리즘의 장점은 분류문제나 예측문제 동시에 쓸 수 있다는 점이다. 또한 정확도 측면에서 다른 분류기 기법들보다 우수한결과를 가져온다고 한다. 하지만 의사결정나무와 같이 직관적인 해석이 불가능하다는 점이 최대 단점이다. 즉 어떤 이유로 데이터들이 분류됐는지를 알 수가 없어 결과에 대한 설명력이 떨어진다고 할 수 있다. 그래도 높은 정확도를 가져오기 때문에 SVM 알고리즘은 널리 사용된다고 한다.

전처리과정에서 설정한 트레이닝 데이터와 테스트 데이터를 기반으로 작업이 이루어진다. 이 트레이닝 데이터를 통해 텍스트 마이닝에서 성능이 가장 우수하다고 평가되는 linear 커널을 이용하여 SVM 모델을 생성한다. 이 모델을 통해 테스트 데이터 셋의 spam 여부를 예측한다. 마지막으로 테스트 데이터와 실제 데이터 셋을 비교하여 SVM 모델 성능평가를 한다.

```
#트레이닝 데이터와 테스트 데이터 분류
svmtrain <- cleandata[Train,]
svmtest <- cleandata[-Train,]

#"e1071" 패키지를 불러와서 svm 을 실행한다. 여기서 scale 은 함수 자체에서 평균 0, 표준편차 1 로 해주는데 필요가 없을 시 False 를 입력한다.
#커널은 텍스트마이닝에 가장효율적이라고 평가되는 linear 모델로 선정
(SVM 모델 트레이닝데이터)
library(e1071)
svm.linear <- svm(Class~., data=svmtrain, scale=FALSE, kernel='linear')

#테스트 데이터 셋의 spam 여부를 predict()함수를 통해 예측한다.
pred.linear <- predict(svm.linear, svmtest[,-1])

#모델의 정확도 확인을 위해 테스트 데이터셋과 실제 데이터셋을 비교
library(gmodels)
CrossTable(pred.linear,svmtest$Class)
```

4. 분석 결과

4.1.1. K-Means

coco

Filter

	1	2	3	4	5	6	7	8	9	10	11	12
1	0.0000000	0.89952448	9.708447e-01	9.852228e-01	1.00000000	0.93844555	0.7038874	0.6234944	0.9740171	0.8066479	0.69963474	0.48
2	0.8995245	0.00000000	8.547529e-01	8.910534e-01	0.84003342	0.85086907	0.4222559	0.8992477	0.8704754	0.8975326	0.63524019	0.72
3	0.9708447	0.85475294	0.000000e+00	1.000000e+00	0.90299243	0.60300566	1.0000000	0.9230928	0.9628029	0.8345861	0.80755825	0.92
4	0.9852228	0.89105340	1.000000e+00	0.000000e+00	0.91546972	0.98026057	0.9369778	0.9805099	0.9768388	0.9844124	0.89169341	0.91
5	1.0000000	0.84003342	9.029924e-01	9.154697e-01	0.00000000	1.00000000	0.9413427	1.0000000	0.9687709	0.9894913	0.84097316	0.93
6	0.9384455	0.85086907	6.030057e-01	9.802606e-01	1.00000000	0.00000000	1.0000000	0.8668354	0.9207699	0.8225542	0.83361015	0.82
7	0.7038874	0.42225591	1.000000e+00	9.369778e-01	0.94134272	1.00000000	0.0000000	0.5936826	1.0000000	1.0000000	0.88204585	0.90
8	0.6234944	0.89924772	9.230928e-01	9.805099e-01	1.00000000	0.86683535	0.5936826	0.0000000	0.9657305	0.7244601	0.62208527	0.42
9	0.9740171	0.87047535	9.628029e-01	9.768388e-01	0.96877094	0.92076995	1.0000000	0.9657305	0.0000000	0.9175821	0.92616403	0.92
10	0.8066479	0.89753256	8.345861e-01	9.844124e-01	0.98949133	0.82255418	1.0000000	0.7244601	0.9175821	0.0000000	0.90945756	0.69
11	0.6996347	0.63524019	8.075582e-01	8.916934e-01	0.84097316	0.83361015	0.8820459	0.6220853	0.9261640	0.9094576	0.00000000	0.49
12	0.4832774	0.72426007	9.225321e-01	9.126884e-01	0.93700803	0.82812605	0.9049110	0.4290525	0.9212005	0.6926503	0.49265908	0.00
13	0.5281853	0.90495099	6.714294e-01	9.845683e-01	0.95914759	0.76711486	1.0000000	0.7272168	0.9728664	0.3802431	0.69615108	0.50
14	0.9619273	0.82426836	8.657335e-01	9.660620e-01	0.34067449	0.94914251	1.0000000	0.9497850	0.8971030	0.9452946	0.82530072	0.94
15	0.7964074	0.91959622	7.950304e-01	9.821503e-01	0.95274636	0.87275995	1.0000000	0.7314767	0.9686148	0.7852426	0.92006856	0.72
16	0.9009015	0.81419774	6.091940e-01	9.805683e-01	1.00000000	0.03559824	0.9523665	0.8689111	0.9454840	0.8411217	0.77274985	0.79
17	0.9801861	0.90854507	9.301245e-01	9.823379e-01	0.65687132	0.80286005	1.0000000	0.9738669	0.9464500	0.9715300	0.90908234	0.97
18	0.9676914	0.49807219	8.019821e-01	1.000000e+00	1.00000000	0.96865683	0.6544549	1.0000000	0.9632236	0.9752493	0.94675542	0.92
19	0.8244481	0.40229046	7.279819e-01	8.469077e-01	0.77521417	0.69863594	0.8332708	0.7677776	0.8179913	0.8197647	0.37417658	0.52
20	0.9487604	0.05268534	9.184600e-01	9.108535e-01	0.84857030	0.95029149	0.3901244	1.0000000	0.9416748	0.9607468	0.76516037	0.81
21	0.2695003	0.76920293	9.632492e-01	9.171587e-01	0.94023322	0.88660323	0.5821847	0.7780639	0.9252350	0.9300684	0.78408967	0.79
22	0.9023561	0.69812446	4.092278e-01	1.000000e+00	0.91847097	0.55370324	0.9398298	0.8707282	0.9133749	0.6183955	0.67734697	0.75
23	0.9018633	0.79140920	6.129872e-01	9.807569e-01	1.00000000	0.02514268	0.9528288	0.8701835	0.9227620	0.8270156	0.77495554	0.78
24	0.9018633	0.79140920	6.129872e-01	9.807569e-01	1.00000000	0.02514268	0.9528288	0.8701835	0.9227620	0.8270156	0.77495554	0.78
25	0.5466372	0.84337592	5.558738e-01	9.815470e-01	0.93870874	0.67230557	0.6754656	0.5696739	0.9440521	0.8542209	0.36133062	0.71
26	0.6121037	0.75340959	8.411405e-01	9.004806e-01	0.86896139	0.84710981	0.5529211	0.4781629	0.9606542	0.9263937	0.43596043	0.59
27	0.9676914	0.49807219	8.019821e-01	1.000000e+00	1.00000000	0.96865683	0.6544549	1.0000000	0.9632236	0.9752493	0.94675542	0.92
28	0.9740171	0.87047535	9.628029e-01	9.768388e-01	0.96877094	0.92076995	1.0000000	0.9657305	0.0000000	0.9175821	0.92616403	0.92
29	0.9638982	0.75659517	8.726839e-01	9.678189e-01	0.37480495	0.88991426	1.0000000	0.9523844	0.8298453	0.8992768	0.83434415	0.89

Showing 1 to 31 of 233 entries

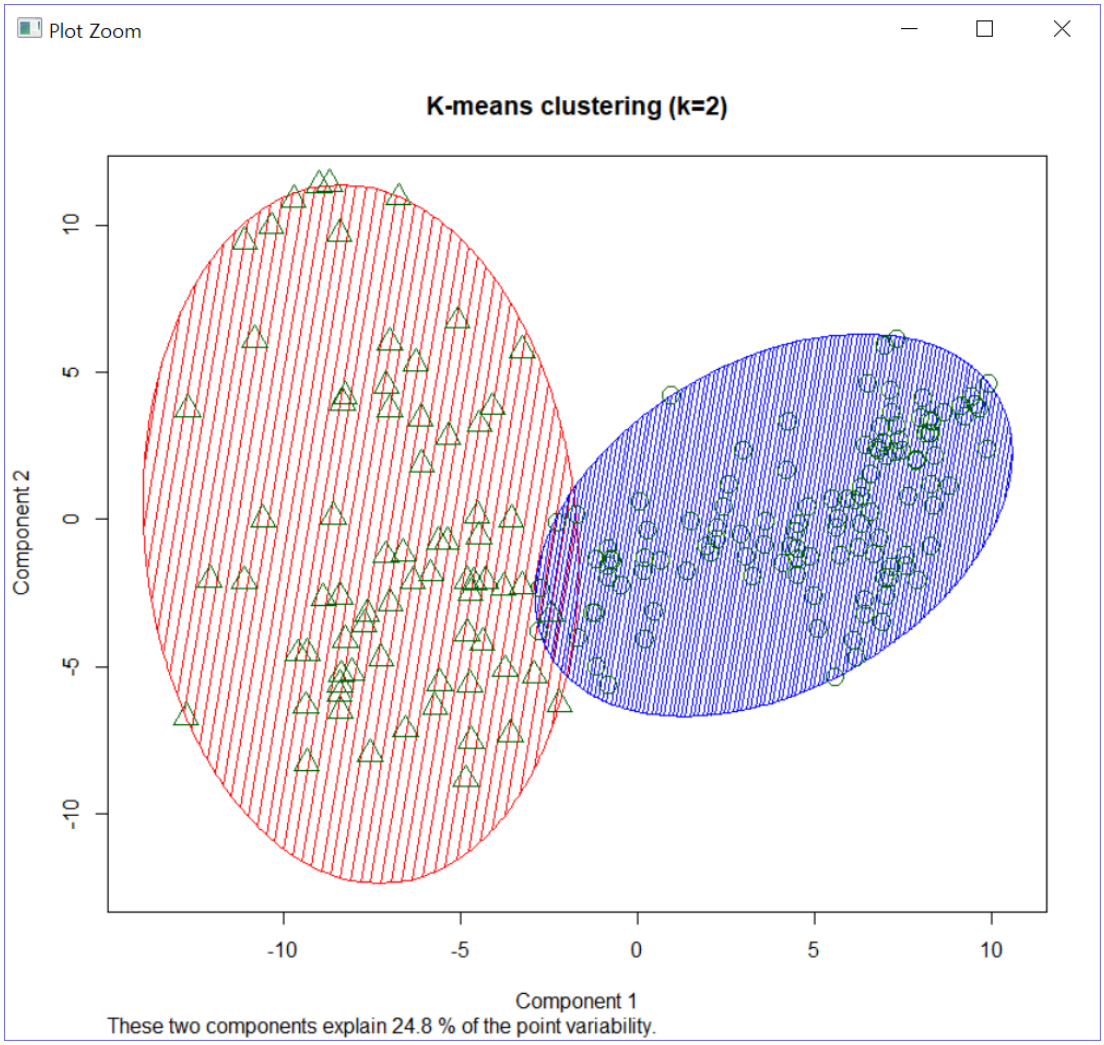
[그림 8. Cosine 방식으로 계산된 TF-IDF 데이터 coco]

위의 그림 8 은 cleandtm 을 Document - Document 형태의 매트릭스로 변환한 파일인 coco 를 나타낸 것이다. Document - Document 형태의 매트릭스를 만들 때, 거리 계산 방법은 Cosine 방식을 사용했고, 그 방식의 거리계산법은 1-(Cosine similarity) 이다.

k = 2 로 클러스터링 했을 때의 결과이다. 아래와 같이 분류되었다.

```
> kmeans_test$cluster
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31
  2   2   2   1   1   2   2   2   1   2   2   2   2   1   1   2   1   1   2   2   2   2   2   2   2   2   1   1   1   1   1
 32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62
  2   2   2   1   2   1   2   2   1   2   1   2   2   2   1   2   1   1   2   1   1   1   2   1   2   1   2   2   2   2   1
 63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93
  2   1   1   2   1   1   2   2   1   2   2   2   1   2   1   1   2   2   1   2   2   1   1   2   1   1   1   1   2   1   1
 94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124
  1   2   1   1   2   2   1   2   2   2   2   2   2   2   2   2   2   1   2   1   2   1   1   2   2   2   1   1   2   1   2
125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155
  2   2   2   2   2   2   2   1   1   2   2   2   2   2   2   2   2   1   1   1   1   2   1   2   1   1   1   2   1   1   1
156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

위의 k = 2 클러스터링 결과를 시각적으로 보기 위해 plot 을 그렸다.



[그림 9. K=2 로 분류된 K-Means 클러스터링 결과 플롯]

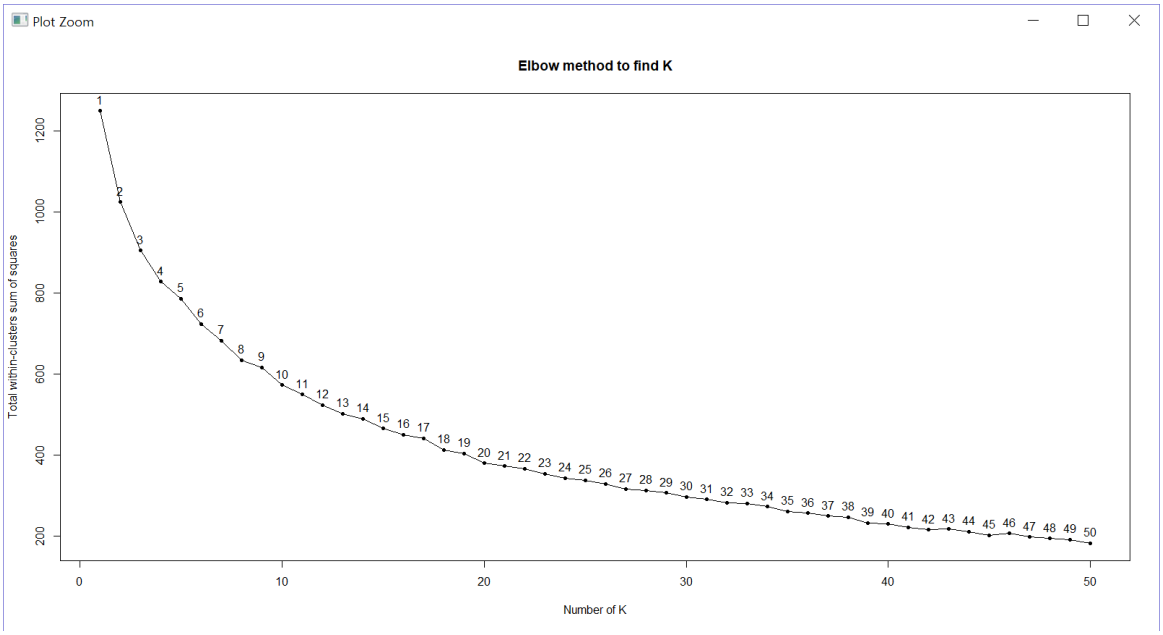
아래는 클러스터링 시 실제 ham document 와 spam document 가 어떻게 군집 되었는지 비교해 보기 위해 그린 table 이다.

```
> table(cleandata$Class,kmeans_test$cluster)

       1  2
ham   80  0
spam  63 90
```

Ham document 의 데이터는 모두 같은 군집인 cluster1 로 분류되었다. 하지만 spam 의 데이터는 약 2:3 의 비율로 2 개의 클러스터에 나뉘어 분류되었다.

Elbow Method 를 사용해 최적의 k 값을 찾아보았다. elbow 는 k 가 42 일 때로 추정한다.



[그림 10. K 값에 따른 K-Means 의 Elbow Method]

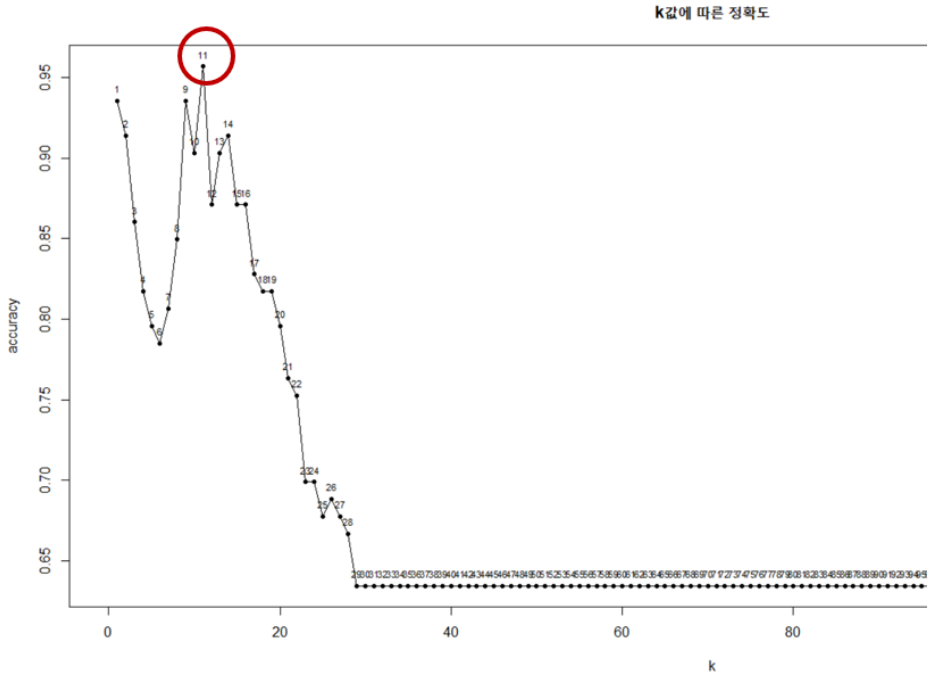
k 의 값을 42 로 설정하고 다시 kmeas 를 실행하고 실제 데이터와 비교해 보았을 때의 결과는 아래 그림 10-1 과 같다. k 가 42 일때도 마찬가지로 모든 클러스터 내의 불순도가 0 이 되지는 않는다.

```
> table(cleandata$Class,kmeans_test$cluster)

      1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
ham   5  5  0  0  0  1  0  0  0  0 11  0  2  0  5  5  0  0  5  5  2  1  4  0  1 10  3  0  0  0  7  0  0  1  0  2  0  0  0  0  5  0
spam  0  0  4  2  8  2  6  6  6  5  0  5  2  4  0  0  3  7  0  0  1  3  0 15  3  0  0  3  7  6  0  4  5  4 10  7  4  6  2  4  0  9
>
```

[그림 10-1. K=42 에 따른 K-Means 의 결과(datatable)]

4.1.2. Knn(k Nearest Neighbor)



[그림 11. Knn 의 k 값에 따른 정확도 plot]

앞서 k 값의 변화의 따른 정확도를 계산 했을 때, k 의 값이 11 인 경우가 가장 정확도가 높았다. 그래서 k=11 로 설정하고 knn 을 실행했을 때
P(knn_ham|test_ham)*P(test_ham)=0.355,
P(knn_spam|test_spam)*P(test_spam)=0.602 를 더하여 0.957, 즉 95.7%의 정확도를 보였다.

```
> CrossTable(test_label,knntest,prop.chisq = F)

   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|

Total Observations in Table:  93

             | knntest
  test_label |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |        33 |         1 |        34 |
             |     0.971 |     0.029 |     0.366 |
             |     0.917 |     0.018 |           |
             |     0.355 |     0.011 |           |
-------------|-----------|-----------|-----------|
        spam |         3 |        56 |        59 |
             |     0.051 |     0.949 |     0.634 |
             |     0.083 |     0.982 |           |
             |     0.032 |     0.602 |           |
-------------|-----------|-----------|-----------|
Column Total |        36 |        57 |        93 |
             |     0.387 |     0.613 |           |
-------------|-----------|-----------|-----------|
```

[그림 12. k=11 일 때 knn 의 분류 결과]

4.1.3. Naïve Bayes

Laplace smoothing 을 하지 않았을 때의 결과이다.
P(pred_ham|actual_ham)*P(actual_ham)=0.355,
P(pred_spam|acutal_spam)*P(acutal_spam)=0.602 를 더하여 95.7%의 정확도를 가진다.

```
> CrossTable(sms_test_pred, naive_test_label,
+            dnn = c('predicted', 'actual'))


   Cell Contents
|-------------------------|
|                       N |
| Chi-square contribution |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  93


              | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |        33 |         3 |        36 |
             |    29.904 |    17.233 |           |
             |     0.917 |     0.083 |     0.387 |
             |     0.971 |     0.051 |           |
             |     0.355 |     0.032 |           |
-------------|-----------|-----------|-----------|
        spam |         1 |        56 |        57 |
             |    18.887 |    10.884 |           |
             |     0.018 |     0.982 |     0.613 |
             |     0.029 |     0.949 |           |
             |     0.011 |     0.602 |           |
-------------|-----------|-----------|-----------|
Column Total |        34 |        59 |        93 |
             |     0.366 |     0.634 |           |
-------------|-----------|-----------|-----------|
```

[그림 13. Laplace smoothing 을 적용하지 않았을 때 Naïve Bayes 분류 결과]

아래는 Laplace smoothing 을 했을 때의 결과이다.
P(pred_ham|actual_ham)*P(actual_ham)=0.355,
P(pred_spam|acutal_spam)*P(acutal_spam)=0.581 를 더하여 93.6%의 정확도를 가진다.

```
> CrossTable(sms_test_pred2, naive_test_label,
+            dnn = c('predicted', 'actual'))


   Cell Contents
|-------------------------|
|                       N |
| Chi-square contribution |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  93


              | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |        33 |         5 |        38 |
             |    26.280 |    15.145 |           |
             |     0.868 |     0.132 |     0.409 |
             |     0.971 |     0.085 |           |
             |     0.355 |     0.054 |           |
-------------|-----------|-----------|-----------|
        spam |         1 |        54 |        55 |
             |    18.157 |    10.464 |           |
             |     0.018 |     0.982 |     0.591 |
             |     0.029 |     0.915 |           |
             |     0.011 |     0.581 |           |
-------------|-----------|-----------|-----------|
Column Total |        34 |        59 |        93 |
             |     0.366 |     0.634 |           |
-------------|-----------|-----------|-----------|
```

[그림 14. Laplace smoothing 을 적용했을 때의 Naïve Bayes 분류 결과]

4.1.4. SVM(Support Vector Machine)

SVM 을 이용해 얻은 결과이다. P(pred_ham|test_ham)*P(test_ham)=0.323, P(pred_spam|test_spam)*P(test_spam)=0.624 를 더하여 94.7%의 정확도를 가진다.

```
> CrossTable(pred.linear,svmtest$Class)


   Cell Contents
|-------------------------|
|                       N |
| Chi-square contribution |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  93


             | svmtest$Class
 pred.linear |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |        30 |         1 |        31 |
             |    30.745 |    17.718 |           |
             |     0.968 |     0.032 |     0.333 |
             |     0.882 |     0.017 |           |
             |     0.323 |     0.011 |           |
-------------|-----------|-----------|-----------|
        spam |         4 |        58 |        62 |
             |    15.373 |     8.859 |           |
             |     0.065 |     0.935 |     0.667 |
             |     0.118 |     0.983 |           |
             |     0.043 |     0.624 |           |
-------------|-----------|-----------|-----------|
Column Total |        34 |        59 |        93 |
             |     0.366 |     0.634 |           |
-------------|-----------|-----------|-----------|
```

[그림 15. SVM 의 분류 결과]

5. 결론 및 한계

5.1.1. 결론

K-means 는 비지도 학습인만큼 label 을 지정해 주지 않았기 때문에, 애초에 군집화가 잘 이루어질 것이라는 기대는 없었다. elbow method 를 통해 구한 k 값 42 에서도 모든 클러스터내의 불순도가 모두 0 이 되지는 않아서 스팸 필터링에는 적절치 않다고 본다. 이 점이 비지도 학습의 한계인 것 같다.

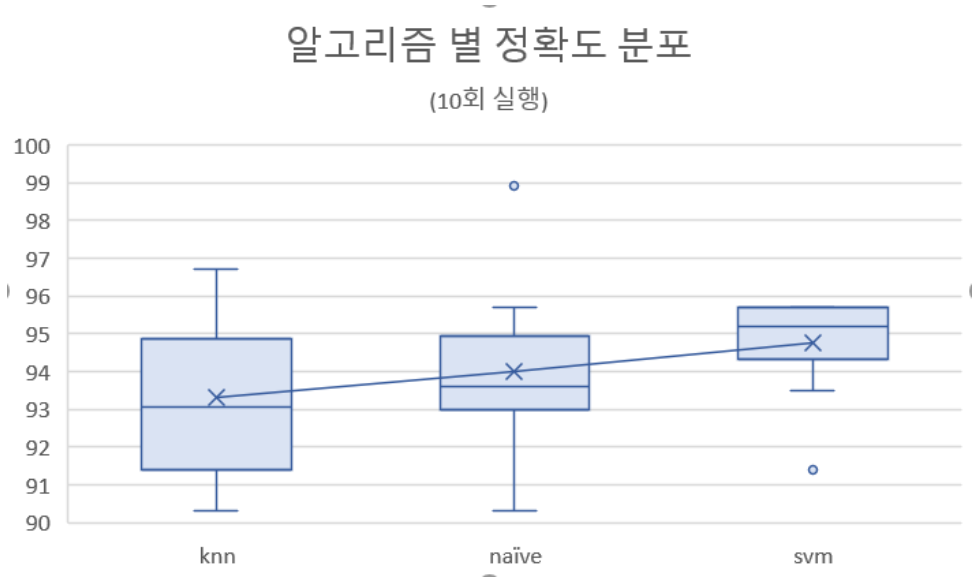
K-means 를 제외한 나머지 세개의 알고리즘은 랜덤 트레이닝 데이터셋을 총 10 개를 뽑아 놓고, Knn, Naïve Bayes, SVM 3 개의 기법을 각 회차마다 성능평가 실시 후 정확도에 따라 순위를 매겼다.

10 회차가 끝난 후 각 알고리즘별 순위를 모두 합해 그 값이 가장 작은 것은 SVM 이 나왔고, 그 다음으로 Naïve bayes, Knn 순으로 나왔다.

횟수	알고리즘	정확도	rank	RANK
1	knn	95.7	1	1
1	naïve	95.7	1	1
1	svm	94.7	3	3
2	knn	90.3	3	3
2	naïve	94.7	2	2
2	svm	95.7	1	1

행 레이블	합계 : RANK2
knn	23
naïve	20
svm	15
총합계	**58**

[그림 16. 효율에 따른 랭크 결과와 그 합계]



[그림 17. 효율에 따른 랭킹결과로 나타낸 플롯]

[그림 17]을 보면 각 박스플롯을 잇는 선은 각 알고리즘의 평균값을 이은 선이다. 이것으로 보아 가장 높은 평균을 갖고 있는 것은 SVM 이며, 세 박스플롯을 비교하였을

때, 가장 편차가 적은 알고리즘도 SVM 이다. 따라서 SVM 이 택배 스미싱 데이터를 필터링 하는 데이 있어서 가장 변동이 적고 성능이 좋다는 것을 알 수 있다.

결론적으로 택배 스미싱 데이터를 필터링하기에 효과적인 알고리즘은 SVM, Naïve Bayes, Knn 순이고, K-Means 는 적합하지 않다는 결론이 나왔다. 더 나아가 지도학습 알고리즘의 경우, 전체적으로 90% 이상의 좋은 분류 결과를 보여주었으므로 SVM 뿐만이 아닌 다른 알고리즘에 대해서도 좋은 결과를 기대할 수 있을 것이다.

5.1.2. 프로젝트의 한계

연구 데이터가 텍스트보다 이미지로 올라와 있는 경우가 대다수였다. 이것은 우리가 데이터를 수집하는데 오랜 시간이 걸리게 만들었다. 또한 택배사 사칭 스미싱 데이터는 개인의 핸드폰으로 발송되는 문자이므로 개인이 사례를 올리거나, 연구자 본인이 받은 것이 아니면 데이터를 수집하기 어려웠다. 따라서 수집한 데이터의 양이 약 230 개 정도로 적은 편이어서 알고리즘이 사용한 데이터에 대해 overfitting 의 가능성이 큰 것이 이 프로젝트의 한계였다.

6. 의의 및 추후 개선 방향

이 연구는 스팸 문자 중 가장 많이 보내지는 택배사 스미싱 문자를 높은 확률로 분류해낼 수 있는 알고리즘을 찾고, 코드로 구현했다는 점에 가장 큰 의의가 있다. 또한 어떤 알고리즘이 가장 효율적이었는지 알아낸 점도 가치 있다고 생각한다.

한계점을 극복하기 위해 대량의 택배 스미싱 데이터를 수집하기 위해 우리가 찾은 방법은 'ETSsecurity 알약 블로그'이다. ESTsecurity 사의 알약프로그램이 스팸문자 데이터를 수집해 공식 블로그에 주별 스팸 문자 수를 알려준다. 따라서 추후 개발시에는 많은 개인들에게 택배 스미싱 문자 내역을 얻거나, 알약 ESTsecurity 에 데이터를 요청하여 데이터의 수를 늘릴 것이다.

[그림 18. ESTsecurity 사의 알약 블로그 주별 게시글]

레퍼런스

[1] 이지원, 이동훈, & 김인석. (2013). SVM 을 이용한 스미싱 탐지기법. *보안공학연구논문지*, *10*(6), 655-668.

[2] 교묘해지는 택배 스미싱 문자 무심코 눌렀다간..., 매일경제, 2019 년 01 월 09 일 https://www.mk.co.kr/news/society/view/2019/01/18698/

[2] ESTsecurity 알약 블로그, 스미싱게시판 https://blog.alyac.co.kr/search/%EC%8A%A4%EB%AF%B8%EC%8B%B1