Assignment #6

SENTIMENT ANALYSIS ON MOVIE REVIEW DATASET

20190552 손지영

사용 데이터 - **imdb.csv** : 영화 리뷰 사이트인 IMDB로부터 추출한 10,000개의 영화 리뷰 정보 - review: 리뷰 텍스트 - sentiment: “negative” or “positive”

```
library(wordcloud)
library(tm)
library(SnowballC)
library(rsample)
library(randomForest)
library(tidyr)
library(ggplot2)
library(caret)
library(glmnet)
library(rpart)
library(car)
```

1번

모델을 수립하기 전에 데이터의 특성을 분석한다. wordcloud 등의 시각화 도구를 적절히 활용하자.

```
# 데이터 불러오기
data = read.csv('imdb.csv', stringsAsFactors = FALSE)
str(data)
```

```
## 'data.frame':    10000 obs. of  2 variables:
##  $ review   : chr  "One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be h
##  $ sentiment: chr  "positive" "positive" "positive" "negative" ...
```

```
# 범주형 변환
data$sentiment = factor(data$sentiment)

# 긍부정 비율 살펴보기
table(data$sentiment)
```

```
##
## negative positive
##     4972     5028
```

```
# 워드클라우드를 위한 긍부정 분리
df_po = subset(data, sentiment=='positive')
```

```
df_ne = subset(data, sentiment=='negative')

# 워드클라우드 생성
wordcloud(df_po$review, max.words = 40, colors=brewer.pal(8, "Dark2"))
```


```
wordcloud(df_ne$review, max.words = 40, colors=brewer.pal(8, "Dark2"))
```

먼저 두 그림에서 모두 “movie”, “film”의 단어가 공통적으로 큰 부분을 차지하는 것을 확인할 수 있다. 이는 해당 리뷰가 영화 관련 리뷰이기 때문임을 짐작할 수 있다. 다음으로 “the”, “one”, “this”와 같이 일상에서 많이 쓰는 단어 역시 공통으로 큰 부분을 차지하는 것을 알 수 있다. 하지만 “good”이라는 단어는 두 부분에서 모두 공통적으로 많이 등장하지만 “bad”는 부정 리뷰에서 많이 등장한다는 것을 확인할 수 있다. 이를 통해 긍부정 리뷰에 따라 사용이 차이나는 단어가 존재함을 알 수 있다. 하지만 위에서 언급했듯이 보편적으로 많이 사용하는 단어가 더 큰 부분을 차지하기 때문에 이를 줄일만한 방법이 필요함을 알 수 있다.

2번

전체 10,000개의 리뷰 텍스트를 대상으로 corpus를 생성하고 bag-of-words 기법을 적용하기 위해 적절한 preprocessing을 수행해보자.

2.(A) & (B)

강의노트에서 다룬 모든 preprocessing 단계를 순서대로 수행한다. 원 텍스트와 preprocessing 후의 텍스트 사이에 어떤 변화가 있는지 리뷰 텍스트의 예를 들어 비교해보자.

```
# document 생성, corpus 생성
df_corpus = VCorpus(VectorSource(data$review))
text_origin = df_corpus[[969]]$content # 내용을 보려면

# 소문자 변환, 숫자 제거
df_clean = tm_map(df_corpus, content_transformer(tolower))
df_clean = tm_map(df_clean, removeNumbers)
text_lower = df_clean[[969]]$content

# stopword 제거
df_clean = tm_map(df_clean, removeWords, stopwords())
text_stopword = df_clean[[969]]$content

# 문장 부호 제거
df_clean = tm_map(df_clean, removePunctuation)
text_sign = df_clean[[969]]$content

# 어미 제거
df_clean = tm_map(df_clean, stemDocument)
text_mom = df_clean[[969]]$content

# 공백 제거
df_clean = tm_map(df_clean, stripWhitespace)
text_sp = df_clean[[969]]$content


# 원리뷰
print(text_origin)
```

```
## [1] "This is a great film. Touching and strong. The direction is without question breathless. Good work to the
```

```
# 소문자 변환
print(text_lower)
```

```
## [1] "this is a great film. touching and strong. the direction is without question breathless. good work to the
```

```
# stopword 제거
print(text_stopword)
```

```
## [1] "   great film. touching  strong.  direction  without question breathless. good work   team.  feel  sorry
```

```
# 문장 부호 제거
print(text_sign)
```

```
## [1] "   great film touching  strong  direction  without question breathless good work   team  feel  sorry  mar
```

```
# 어미 제거
```

```
print(text_mom)
```

```
## [1] "great film touch strong direct without question breathless good work team feel sorri marlen grace god go"
```

```
# 공백제거
print(text_sp)
```

```
## [1] "great film touch strong direct without question breathless good work team feel sorri marlen grace god go"
```

각각의 전처리 방법을 통해 해당 작업이 잘 적용된 것을 확인할 수 있다.

```
# DTM 생성
df_dtm = DocumentTermMatrix(df_clean)
df_dtm
```

```
## <<DocumentTermMatrix (documents: 10000, terms: 48717)>>
## Non-/sparse entries: 919984/486250016
## Sparsity           : 100%
## Maximal term length: 51
## Weighting          : term frequency (tf)
```

```
#inspect(df_dtm[1:5, 1:10])

# tf-idf 계산
review_tfidf = weightTfIdf(df_dtm)
inspect(review_tfidf[1:10, ])
```

```
## <<DocumentTermMatrix (documents: 10, terms: 48717)>>
## Non-/sparse entries: 669/486501
## Sparsity           : 100%
## Maximal term length: 51
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
## Sample             :
##     Terms
## Docs      camp       gut  halliwel      jake   keitel  laughter    mattei
##   1  0.0000000 0.0000000 0.0000000 0.0000000 0.000000 0.0000000 0.0000000
##   10 0.3154259 0.3695224 0.0000000 0.0000000 0.000000 0.3756824 0.0000000
##   2  0.0000000 0.0000000 0.3126521 0.0000000 0.000000 0.0000000 0.0000000
##   3  0.0000000 0.0000000 0.0000000 0.0000000 0.000000 0.0000000 0.0000000
##   4  0.0000000 0.0000000 0.0000000 0.5603615 0.000000 0.0000000 0.0000000
##   5  0.0000000 0.0000000 0.0000000 0.0000000 0.000000 0.0000000 0.3321928
##   6  0.0000000 0.0000000 0.0000000 0.0000000 0.000000 0.0000000 0.0000000
##   7  0.0000000 0.0000000 0.0000000 0.0000000 0.000000 0.0000000 0.0000000
##   8  0.0000000 0.0000000 0.0000000 0.0000000 0.000000 0.0000000 0.0000000
##   9  0.0000000 0.0000000 0.0000000 0.0000000 0.326747 0.0000000 0.0000000
##     Terms
## Docs       mom   seahunt    wrench
##   1  0.0000000 0.0000000 0.0000000
##   10 0.3315278 0.0000000 0.4519617
##   2  0.0000000 0.0000000 0.0000000
##   3  0.0000000 0.0000000 0.0000000
##   4  0.0000000 0.0000000 0.0000000
##   5  0.0000000 0.0000000 0.0000000
##   6  0.0000000 0.0000000 0.0000000
##   7  0.0000000 0.4218321 0.0000000
##   8  0.0000000 0.0000000 0.0000000
```

```
##   9  0.0000000 0.0000000 0.0000000
```

행렬에 표현된 값을 통해 TF-IDF가 잘 적용된 것을 확인할 수 있다.

```
df_tf_dtm1 = removeSparseTerms(review_tfidf, 0.995)
df_tf = data.frame(as.matrix(df_tf_dtm1))
df_tf$Y = data$sentiment

# train, test set 분리 함수
split_process = function(x, y, z){
  dtm = removeSparseTerms(x, y)
  df_m = data.frame(as.matrix(dtm))
  df_m$Y = z$sentiment
  df_total = df_m[1:3000, ] # 문제에서 주어진대로 train set 분리
  df_test = df_m[3001:10000, ]
  set.seed(123); split = initial_split(df_total, prop=0.8, strata='Y')
  # train set을 8:2로 분리
  df_train = training(split)
  df_val = testing(split)
  return (list(df_train, df_val, df_test))
}
```

반복되는 작업을 피하기 위해 train set과 test을 생성하는 함수를 만들어 준다.

4.1 TF-IDF 효과 비교

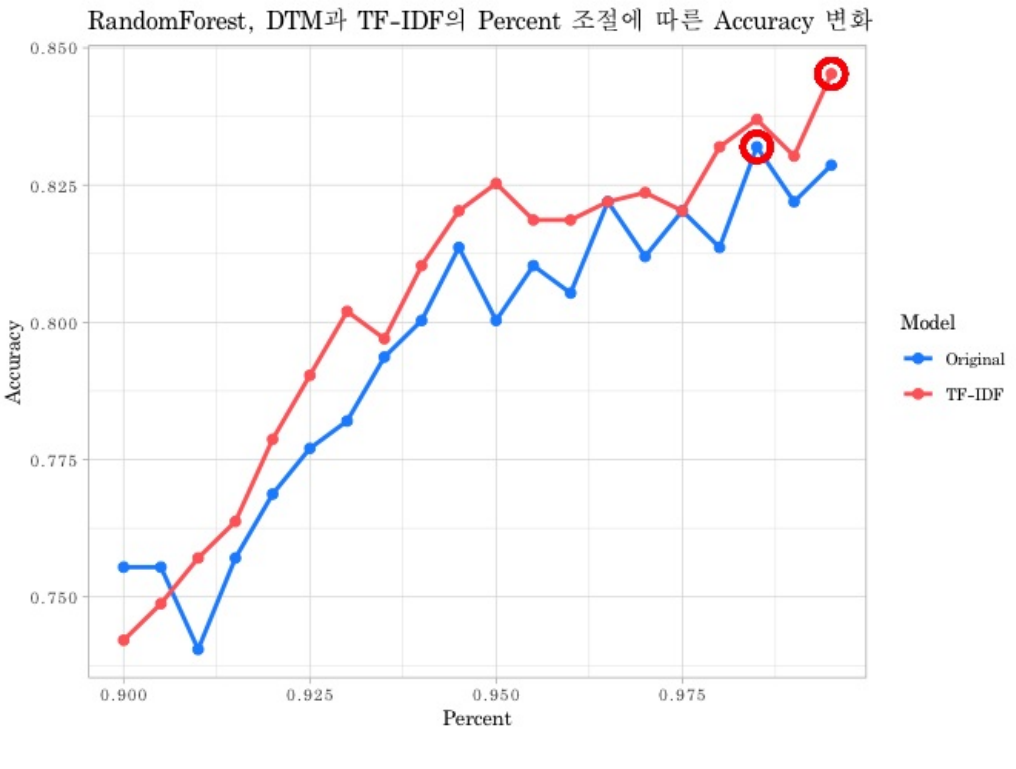
TF-IDF를 적용한 데이터와 아닌 데이터의 성능 차이를 살펴보기 위해 일반화 성능이 좋은 모델인 RandomForest를 통해 비교한다. 추가적으로 몇 % 발생빈도를 가지는 단어를 제외했을때의 성능도 함께 구해 같이 비교해준다.

```
# 결과를 저장하기 위한 빈 데이터 형성
result_origin = data.frame(Percent=numeric(),
                           Accuracy=numeric())

# TF-IDF를 적용하지 않은 데이터 사용
# 각각의 %의 성능을 구하기 위한 반복문 수행
for (i in 1:20){
  num = 1-5*i/1000
  result = split_process(df_dtm, num, data) # 함수를 통한 데이터 생성
  df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
  set.seed(123); rf = randomForest(Y~., data=df_train)
  rf_prob = predict(rf, newdata=df_val, type='prob')
  rf_pred = predict(rf, newdata=df_val, type='class')
  cm = confusionMatrix(rf_pred, df_val$Y, positive='negative')

  # 결과 저장
  result_origin = rbind(result_origin,
                        data.frame(Percent=num,
                                   Accuracy=cm$overall['Accuracy']))
}

# TF-IDF를 적용한 데이터 사용
result_tfidf = data.frame(Percent=numeric(),
                          Accuracy=numeric())
for (i in 1:20){
  num = 1-5*i/1000
  result = split_process(review_tfidf, num, data)
  df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
  set.seed(123); rf = randomForest(Y~., data=df_train) # ntree는 커지면 성능 올라가
  rf_prob = predict(rf, newdata=df_val, type='prob')
  rf_pred = predict(rf, newdata=df_val, type='class')
```

```
  cm = confusionMatrix(rf_pred, df_val$Y, positive='negative')

  result_tfidf = rbind(result_tfidf,
                       data.frame(Percent=num,
                                  Accuracy=cm$overall['Accuracy']))
}

# 두 방법으로 이름 열 추가
result_origin['Model'] = 'Original'
result_tfidf['Model'] = 'TF-IDF'

# 두 데이터프레임을 합친다
df_tfidf = data.frame(); df_tfidf = rbind(result_origin, result_tfidf)

# 각각의 방법에서 최대값과 최소값을 표시하기 위해 해당 값을 따로 구한다
ori_x = result_origin$Percent[which.max(result_origin$Accuracy)]
ori_val = max(result_origin$Accuracy)
tf_x = result_tfidf$Percent[which.max(result_tfidf$Accuracy)]
tf_val = max(result_tfidf$Accuracy)
tf_coef = removeSparseTerms(review_tfidf, 0.995)$ncol
```

```
# 두 방법의 결과값을 표시한다.
ggplot(df_tfidf, aes(x=Percent, y=Accuracy, color=Model)) +
  geom_line(size=1) +
  geom_point(size=2) +
  theme_light() +
  theme(text = element_text(size = 10, family = "AppleMyungjo")) +
  labs(title='RandomForest, DTM과 TF-IDF의 Percent 조절에 따른 Accuracy 변화') +
  geom_point(aes(x = ori_x, y = ori_val, shape = as.factor(ori_x)),
             size = 5, color = "red", shape=1, stroke=2) +
  geom_point(aes(x = tf_x, y = tf_val, shape = as.factor(tf_x)),
             size = 5, color = "red", shape=1, stroke=2) +
  scale_color_manual(values = c("Dodger Blue", "Indian Red1"))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



두 결과를 통해 TF-IDF를 적용한 텍스트의 성능이 거의 모든 퍼센트에서 높다는 것을 알 수 있다. 따라서 다음부터 TF-IDF를 적용한 데이터셋만을 사용하여 모델들의 성능을 비교하도록 하겠다. 또한 두 방법에서 0.93 이후로 지속해서 떨어지기 때문에 다음부터는 0.93~0.995 사이의 퍼센트를 비교하도록 한다.

4.2 Ridge, Lasso 모델 비교

먼저 Ridge, Lasso모델을 구축하기 위해 사용되는 함수를 생성한다.

```
# ridge, lasso model "min" 함수
min_selection_model = function(train_x, train_y, val_x, val_y, alp){
  set.seed(123); cv_model = cv.glmnet(x=train_x, y=train_y, alpha=alp, family='binomial',
                                      type.measure='class', nfolds=10)
  pred = predict(cv_model, s=cv_model$lambda.min, newx=val_x, type='class')
  cm = confusionMatrix(factor(pred), val_y, positive='negative')
  acc = cm$overall['Accuracy']
  num =  sum(coef(cv_model, s = cv_model$lambda.min) != 0)
  return (list(acc, num))
}

# ridge, lasso model "1se" 함수
one_selection_model = function(train_x, train_y, val_x, val_y, alp){
  set.seed(123); cv_model = cv.glmnet(x=train_x, y=train_y, alpha=alp, family='binomial',
                                      type.measure='class', nfolds=10)
  pred = predict(cv_model, s=cv_model$lambda.1se, newx=val_x, type='class')
  cm = confusionMatrix(factor(pred), val_y, positive='negative')
  acc = cm$overall['Accuracy']
  num =  sum(coef(cv_model, s = cv_model$lambda.1se) != 0)
  return (list(acc, num))
}
```

두 함수의 차이점은 첫번째 함수는 CV 수행 후 best model를 사용하는 것이고, 두번째 함수는 CV 수행 후 one-standard-error-rule을 적용하도록 한다.

```
# ridge, min
result_ridge_min = data.frame(Percent=numeric(),
                              Accuracy=numeric())
for (i in 1:14){
  num = 1-5*i/1000
  result = split_process(review_tfidf, num, data)
  df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
  trainx = model.matrix(Y~., data=df_train)[, -1]
  trainy = df_train$Y
  valx = model.matrix(Y~., data=df_val)[, -1]
  valy = df_val$Y
  model_result = min_selection_model(trainx, trainy, valx, valy, 0) # alpha를 0으로 해서 Ridge 모델
  result_ridge_min = rbind(result_ridge_min,
                       data.frame(Percent=num,
                                  Accuracy=model_result[[1]],
                                  Coef_Num = model_result[[2]]))
}


# ridge, 1se
result_ridge_1se = data.frame(Percent=numeric(),
                              Accuracy=numeric())
for (i in 1:14){
  num = 1-5*i/1000
  result = split_process(review_tfidf, num, data)
  df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
  trainx = model.matrix(Y~., data=df_train)[, -1]
```

```
  trainy = df_train$Y
  valx = model.matrix(Y~., data=df_val)[, -1]
  valy = df_val$Y
  model_result = one_selection_model(trainx, trainy, valx, valy, 0)
  result_ridge_1se = rbind(result_ridge_1se,
                           data.frame(Percent=num,
                                      Accuracy=model_result[[1]],
                                      Coef_Num = model_result[[2]]))
}


# lasso, min
result_lasso_min = data.frame(Percent=numeric(),
                              Accuracy=numeric())
for (i in 1:14){
  num = 1-5*i/1000
  result = split_process(review_tfidf, num, data)
  df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
  trainx = model.matrix(Y~., data=df_train)[, -1]
  trainy = df_train$Y
  valx = model.matrix(Y~., data=df_val)[, -1]
  valy = df_val$Y
  model_result = min_selection_model(trainx, trainy, valx, valy, 1) # alpha를 1으로 해서 Lasso 모델
  result_lasso_min = rbind(result_lasso_min,
                           data.frame(Percent=num,
                                      Accuracy=model_result[[1]],
                                      Coef_Num = model_result[[2]]))
}


# lasso, 1se
result_lasso_1se = data.frame(Percent=numeric(),
                              Accuracy=numeric())
for (i in 1:14){
  num = 1-5*i/1000
  result = split_process(review_tfidf, num, data)
  df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
  trainx = model.matrix(Y~., data=df_train)[, -1]
  trainy = df_train$Y
  valx = model.matrix(Y~., data=df_val)[, -1]
  valy = df_val$Y
  model_result = one_selection_model(trainx, trainy, valx, valy, 1)
  result_lasso_1se = rbind(result_lasso_1se,
                           data.frame(Percent=num,
                                      Accuracy=model_result[[1]],
                                      Coef_Num = model_result[[2]]))
}
```

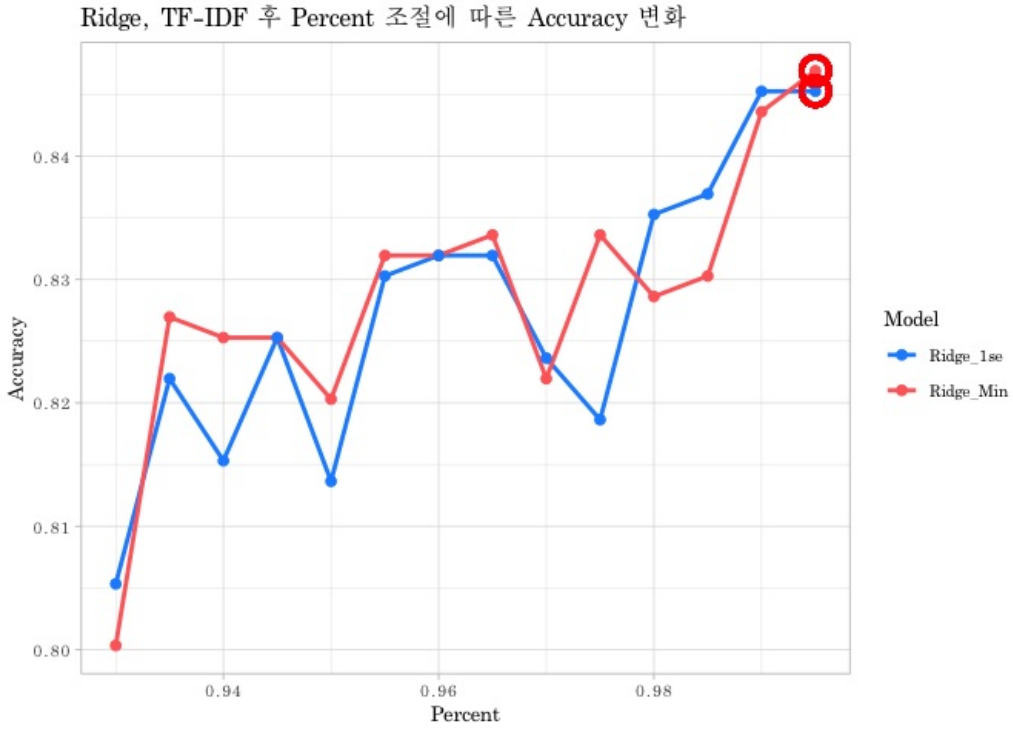
반복문을 통해 0.93~0.995% 사이의 퍼센트에서 각각의 모델의 accuracy를 저장한다.

```
# 각 모델이름과 방법을 나타내는 열을 저장한다.
result_ridge_min['Model'] = 'Ridge_Min'
result_ridge_1se['Model'] = 'Ridge_1se'
result_lasso_min['Model'] = 'Lasso_Min'
result_lasso_1se['Model'] = 'Lasso_1se'

df_ridge = data.frame(); df_ridge = rbind(result_ridge_min, result_ridge_1se)
df_lasso = data.frame(); df_lasso = rbind(result_lasso_min, result_lasso_1se)

# 각각의 방법에서 plot에 표시하기 위한 최고 성능 저장
r_min_x = result_ridge_min$Percent[which.max(result_ridge_min$Accuracy)]
r_min_val = max(result_ridge_min$Accuracy)
r_min_coef = result_ridge_min$Coef_Num[which.max(result_ridge_min$Accuracy)]
r_one_class = result_ridge_1se$Percent[which.max(result_ridge_1se$Accuracy)]
```

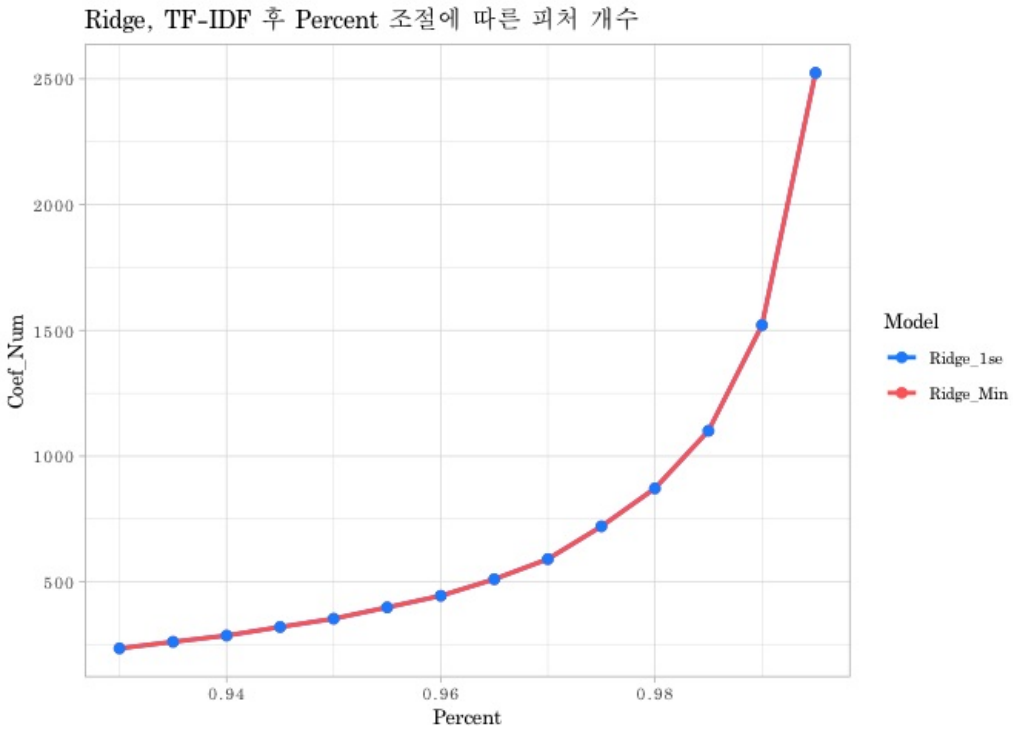
```
r_one_val = max(result_ridge_1se$Accuracy)
r_one_coef = result_ridge_1se$Coef_Num[which.max(result_ridge_1se$Accuracy)]


# Ridege Min모델과 1se 모델의 그림
p1 = ggplot(df_ridge, aes(x=Percent, y=Accuracy, color=Model)) +
  geom_line(size=1) +
  geom_point(size=2) +
  theme_light() +
  labs(title='Ridge, TF-IDF 후 Percent 조절에 따른 Accuracy 변화') +
  geom_point(aes(x = r_min_x, y = r_min_val, shape = as.factor(r_min_x)),
             size = 5, color = "red", shape=1, stroke=2) +
  geom_point(aes(x = r_one_class, y = r_one_val, shape = as.factor(r_one_class)),
             size = 5, color = "red", shape=1, stroke=2) +
  scale_color_manual(values = c("Dodger Blue", "Indian Red1"))

p11 = ggplot(df_ridge, aes(x=Percent, y=Coef_Num, color=Model)) +
  geom_line(size=1) +
  geom_point(size=2) +
  theme_light() +
  labs(title='Ridge, TF-IDF 후 Percent 조절에 따른 피처 개수') +
  scale_color_manual(values = c("Dodger Blue", "Indian Red1"))


l_min_x = result_lasso_min$Percent[which.max(result_lasso_min$Accuracy)]
l_min_val = max(result_lasso_min$Accuracy)
l_min_coef = result_lasso_min$Coef_Num[which.max(result_lasso_min$Accuracy)]
l_one_class = result_lasso_1se$Percent[which.max(result_lasso_1se$Accuracy)]
l_one_val = max(result_lasso_1se$Accuracy)
l_one_coef = result_lasso_1se$Coef_Num[which.max(result_lasso_1se$Accuracy)]


# Lasso Min모델과 1se 모델의 그림
p2 = ggplot(df_lasso, aes(x=Percent, y=Accuracy, color=Model)) +
  geom_line(size=1) +
  geom_point(size=2) +
  theme_light() +
  theme(text = element_text(size = 10, family = "AppleMyungjo")) +
  labs(title='Lasso, TF-IDF 후 Percent 조절에 따른 Accuracy 변화') +
  geom_point(aes(x = l_min_x, y = l_min_val, shape = as.factor(l_min_x)),
             size = 5, color = "red", shape=1, stroke=2) +
  geom_point(aes(x = l_one_class, y = l_one_val, shape = as.factor(l_one_class)),
             size = 5, color = "red", shape=1, stroke=2) +
  scale_color_manual(values = c("Indian Red1", "Dodger Blue"))

p22 = ggplot(df_lasso, aes(x=Percent, y=Coef_Num, color=Model)) +
  geom_line(size=1) +
  geom_point(size=2) +
  theme_light() +
  theme(text = element_text(size = 10, family = "AppleMyungjo")) +
  labs(title='Lasso, TF-IDF 후 Percent 조절에 따른 피처 개수') +
  scale_color_manual(values = c("Dodger Blue", "Indian Red1")) +
    geom_point(aes(x = l_min_x, y = l_min_coef, shape = as.factor(l_min_x)),
             size = 5, color = "red", shape=1, stroke=2) +
  geom_point(aes(x = l_one_class, y = l_one_coef, shape = as.factor(l_one_class)),
             size = 5, color = "red", shape=1, stroke=2)
```

ridge model

```
# Percent에 따른 모델의 성능 비교
p1 + theme(text = element_text(size = 10, family = "AppleMyungjo"))
```



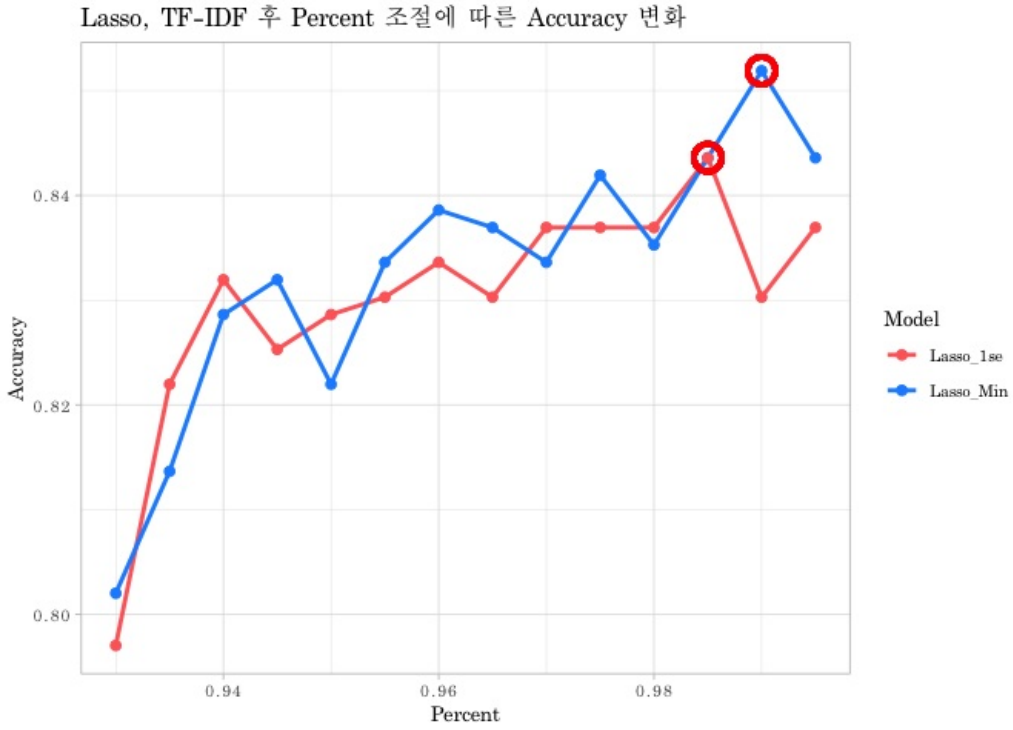

```
# Percent에 따른 사용된 피처 개수
p11+ theme(text = element_text(size = 10, family = "AppleMyungjo"))
```



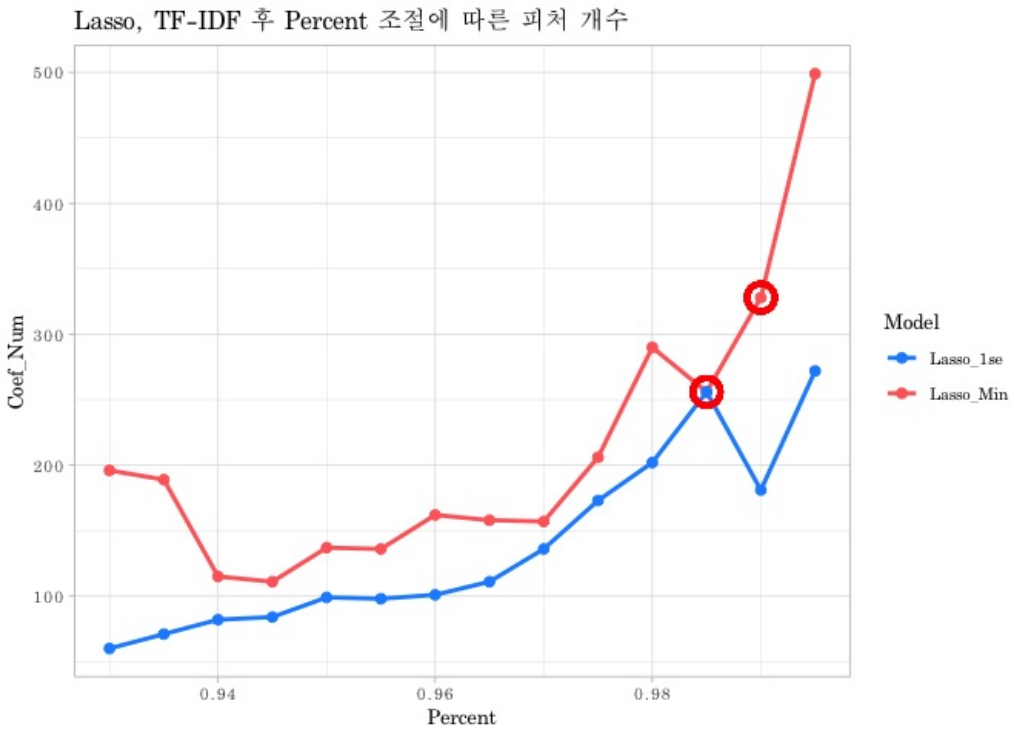
먼저 첫번째 그림을 통해 Ridge 모델에서 best모델을 1se로 했을 때와 min으로 했을 때 모두 0.995에서 성능이 가장 높은 것을 확인할 수 있다. 이는 피처의 수가 많으면 많을수록 예측을 잘 한다는 것을 의미한다. 하지만 피처의 수가 많을수록 과적합될 가능성이 있기 때문에 적절한 수의 피처를 선택해야한다. 해당그래프에서 percent가 줄어듦에 따라 피처 수는 감소하지만 ridge모델 자체가 피처의 수를 감소시켜주는 것은 아니기 때문에 percent가 줄어듦에도 높은 수의 피처들을 갖고 있는 것을 확인할 수 있다.

Lasso model

```
p2+ theme(text = element_text(size = 10, family = "AppleMyungjo"))
```




```
p22+ theme(text = element_text(size = 10, family = "AppleMyungjo"))
```



위의 두 그림을 통해 Lasso모델과 같은 경우 Ridge와 다르게 Percent가 0.99, 0.985일 때 가장 높은 성능임을 확인할 수 있다. 또한 두번째 그림을 통해 해당 percent에서 각 모델이 가지는 피처의 수를 확인해보면 데이터가 1개가 넘었던 ridge와 다르게 변수 selection 효과로 500개 미만인 것을 확인할 수 있다. 또한 one-standar-rule을 적용한 파란색 그래프가 상대적으로 더 적은 피처수를 갖고 있는 것을 확인할 수 있다. 또한 Ridge모델보다 훨씬 더 적은 변수를 사용했지만 성능 차이가 크지 않은 것을 보아 과적합에 훨씬 더 잘 대응할 수 있을 것이라 예상된다.

ElasticNet

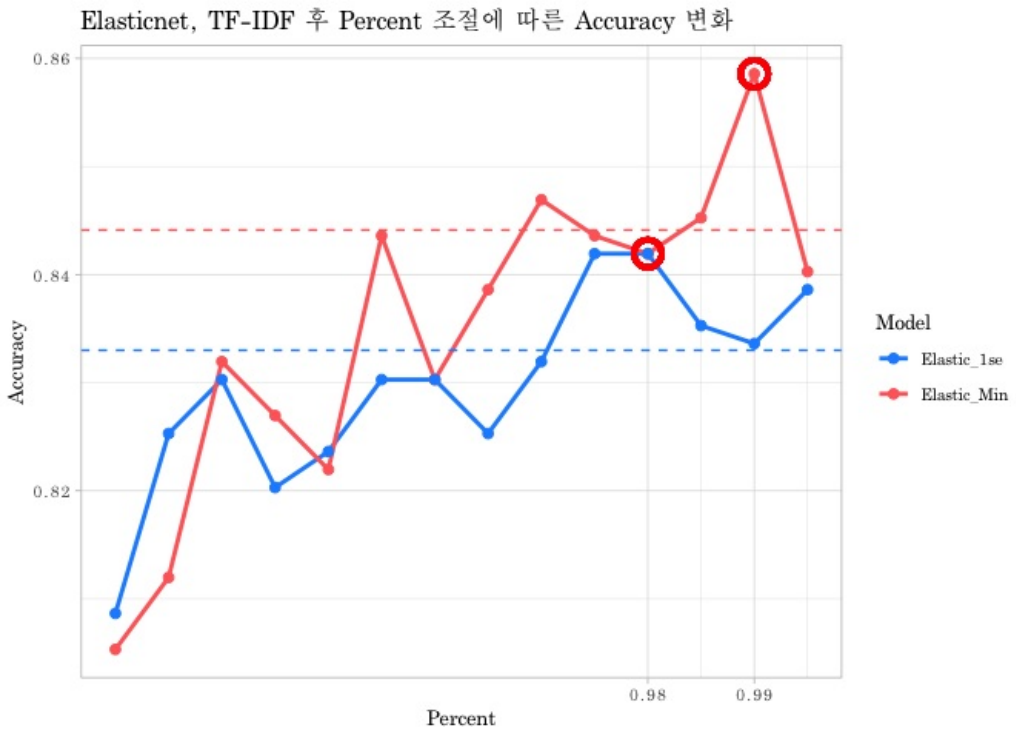
Elasticnet은 선형 회귀 모델의 일종으로 L1 규제와 L2 규제를 함께 사용하여 변수 선택과 모델의 복잡성 조절을 동시에 수행하는 방법이다. 따라서 Lasso모델의 변수 선택 능력과 Ridge 모델의 안정성을 모두 가질 수 있다는 장점이 있다. 해당 모델의 성능의 보편적으로 더 좋다하여 해당 모델을 사용하여 예측을 진행해보았다.

```
# ElasticNet, min
num_coef_min = data.frame(num=numeric())
result_ela_min = data.frame(Percent=numeric(),
                            Accuracy=numeric())
for (i in 1:14){
  num = 1-5*i/1000
  result = split_process(review_tfidf, num, data)
  df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
  trainx = model.matrix(Y~., data=df_train)[, -1]
  trainy = df_train$Y
  valx = model.matrix(Y~., data=df_val)[, -1]
  valy = df_val$Y
  model_result = min_selection_model(trainx, trainy, valx, valy, 0.5)
  result_ela_min = rbind(result_ela_min,
                         data.frame(Percent=num,
                                    Accuracy=model_result[[1]],
                                    Coef_Num = model_result[[2]]))
}


# ElasticNet, 1se
result_ela_1se = data.frame(Percent=numeric(),
                            Accuracy=numeric())
for (i in 1:14){
  num = 1-5*i/1000
  result = split_process(review_tfidf, num, data)
  df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
  trainx = model.matrix(Y~., data=df_train)[, -1]
  trainy = df_train$Y
  valx = model.matrix(Y~., data=df_val)[, -1]
  valy = df_val$Y
  model_result = one_selection_model(trainx, trainy, valx, valy, 0.5)
  result_ela_1se = rbind(result_ela_1se,
                         data.frame(Percent=num,
                                    Accuracy=model_result[[1]],
                                    Coef_Num = model_result[[2]]))
}
```

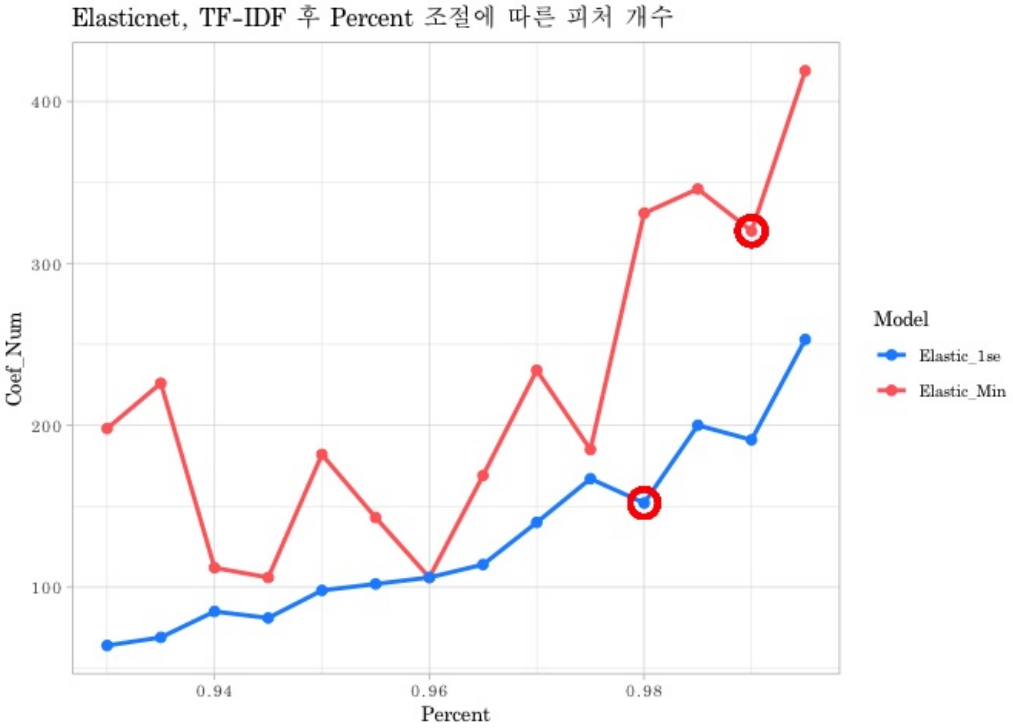
```
result_ela_min['Model'] = 'Elastic_Min'
result_ela_1se['Model'] = 'Elastic_1se'


df_ela = data.frame(); df_ela = rbind(result_ela_min, result_ela_1se)

ela_min_x = result_ela_min$Percent[which.max(result_ela_min$Accuracy)]
ela_min_coef = result_ela_min$Coef_Num[which.max(result_ela_min$Accuracy)]
ela_min_val = max(result_ela_min$Accuracy)
ela_one_class = result_ela_1se$Percent[which.max(result_ela_1se$Accuracy)]
ela_one_coef = result_ela_1se$Coef_Num[which.max(result_ela_1se$Accuracy)]
ela_one_val = max(result_ela_1se$Accuracy)

ela_min_thre = result_ela_min[which.max(result_ela_min$Accuracy), ]
ela_min_thre = ela_min_thre$Accuracy - sd(result_ela_min$Accuracy)
ela_one_thre = result_ela_1se[which.max(result_ela_1se$Accuracy), ]
ela_one_thre = ela_one_thre$Accuracy - sd(result_ela_1se$Accuracy)


p_ela1 = ggplot(df_ela, aes(x=Percent, y=Accuracy, color=Model)) +
  geom_line(size=1) +
  geom_point(size=2) +
  theme_light() +
  labs(title='Elasticnet, TF-IDF 후 Percent 조절에 따른 Accuracy 변화') +
  geom_point(aes(x = ela_min_x, y = ela_min_val, shape = as.factor(ela_min_x)),
             size = 5, color = "red", shape=1, stroke=2) +
  geom_point(aes(x = ela_one_class, y = ela_one_val, shape = as.factor(ela_one_class)),
```

```
            size = 5, color = "red", shape=1, stroke=2) +
  scale_color_manual(values = c("Dodger Blue", "Indian Red1")) +
  geom_hline(yintercept = ela_min_thre, linetype = "dashed", color = "Indian Red1") +
  geom_hline(yintercept = ela_one_thre, linetype = "dashed", color = "Dodger Blue") +
  scale_x_continuous(breaks=c(ela_min_x, ela_one_class))

p_ela2 = ggplot(df_ela, aes(x=Percent, y=Coef_Num, color=Model)) +
  geom_line(size=1) +
  geom_point(size=2) +
  theme_light() +
  labs(title='Elasticnet, TF-IDF 후 Percent 조절에 따른 피처 개수') +
  scale_color_manual(values = c("Dodger Blue", "Indian Red1")) +
    geom_point(aes(x = ela_min_x, y = ela_min_coef, shape = as.factor(ela_min_x)),
            size = 5, color = "red", shape=1, stroke=2) +
  geom_point(aes(x = ela_one_class, y = ela_one_coef, shape = as.factor(ela_one_class)),
            size = 5, color = "red", shape=1, stroke=2)
```

```
p_ela1+ theme(text = element_text(size = 10, family = "AppleMyungjo"))
```




```
p_ela2+ theme(text = element_text(size = 10, family = "AppleMyungjo"))
```



먼저 첫번째 그림을 통해 Elastic model은 0.99, 0.98 퍼센트의 경우에 가장 높은 성능을 갖음을 확인할 수 있다. 또한 변수의 개수를 비교해보면 Lasso와 동일하게 모두 500개 아래로 적은 피처 수를 가지고 있음을 알 수 있다. Elastic 모델 역시 1se를 사용했을때 더 적은 피처를 갖게 되는데 최고 성능에서 190정도의 피처를 갖고 0.84 정도의 높은 성능을 보인다.

추가적으로, alpha값을 조절하면 Elastic모델에서 L2와 L1 규제 정도를 조절할 수 있다하여 alpha값을 조절하며 결과를 비교해보고자 한다. 따라서 위의 그래프 결과를 토대로 Elastic min모델에서는 0.99 퍼센트로 고정하고, Elastic 1se모델에서는 0.98로 조정하여 베스트 모델을 찾을 수 있도록 비교한다.

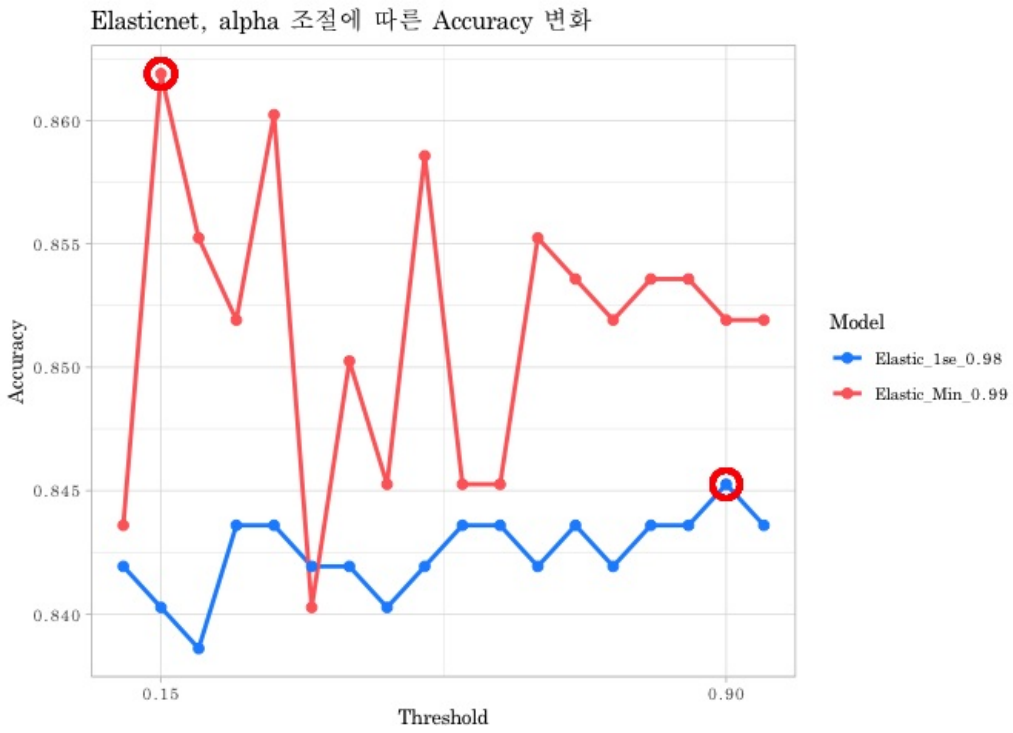
Elastic net alpha 조절

```
# 0.99, ElasticNet, min
result_thre_ela_min = data.frame(Percent=numeric(),
                                 Accuracy=numeric())
for (i in 2:19){
  num = i/20
  result = split_process(review_tfidf, 0.99, data)
  df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
  trainx = model.matrix(Y~., data=df_train)[, -1]
  trainy = df_train$Y
  valx = model.matrix(Y~., data=df_val)[, -1]
  valy = df_val$Y
  model_result = min_selection_model(trainx, trainy, valx, valy, num)
  result_thre_ela_min = rbind(result_thre_ela_min,
                          data.frame(Percent=num,
                                     Accuracy=model_result[[1]],
                                     Coef_Num = model_result[[2]]))
}

# 0.98, ElasticNet, 1se
result_thre_ela_1se = data.frame(Percent=numeric(),
                                 Accuracy=numeric())
for (i in 2:19){
  num = i/20
  result = split_process(review_tfidf, 0.98, data)
  df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
  trainx = model.matrix(Y~., data=df_train)[, -1]
  trainy = df_train$Y
  valx = model.matrix(Y~., data=df_val)[, -1]
  valy = df_val$Y
  model_result = one_selection_model(trainx, trainy, valx, valy, num)
```

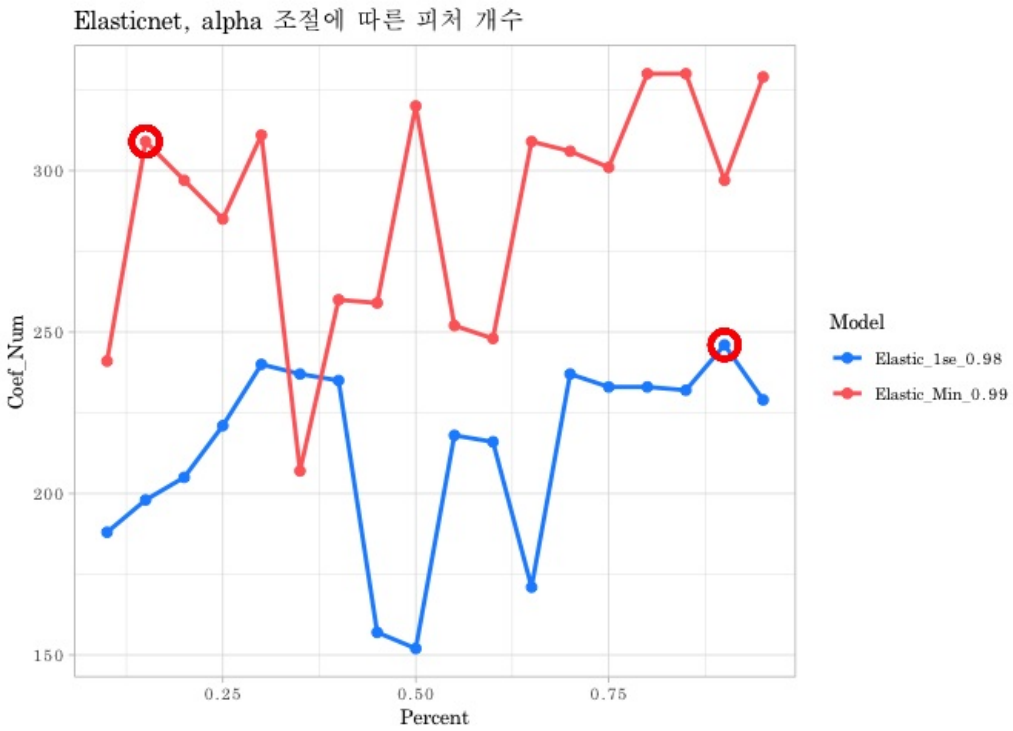
```
  result_thre_ela_1se = rbind(result_thre_ela_1se,
                              data.frame(Percent=num,
                                         Accuracy=model_result[[1]],
                                         Coef_Num = model_result[[2]]))
}
```

```
result_thre_ela_min['Model'] = 'Elastic_Min_0.99'
result_thre_ela_1se['Model'] = 'Elastic_1se_0.98'


df_thre_ela = data.frame(); df_thre_ela = rbind(result_thre_ela_min, result_thre_ela_1se)

ela_thre_min_x = result_thre_ela_min$Percent[which.max(result_thre_ela_min$Accuracy)]
ela_thre_min_coef = result_thre_ela_min$Coef_Num[which.max(result_thre_ela_min$Accuracy)]
ela_thre_min_val = max(result_thre_ela_min$Accuracy)
ela_thre_one_class = result_thre_ela_1se$Percent[which.max(result_thre_ela_1se$Accuracy)]
ela_thre_one_coef = result_thre_ela_1se$Coef_Num[which.max(result_thre_ela_1se$Accuracy)]
ela_thre_one_val = max(result_thre_ela_1se$Accuracy)


ela_thr = ggplot(df_thre_ela, aes(x=Percent, y=Accuracy, color=Model)) +
  geom_line(size=1) +
  geom_point(size=2) +
  theme_light() +
  labs(title='Elasticnet, alpha 조절에 따른 Accuracy 변화') +
  geom_point(aes(x = ela_thre_min_x, y = ela_thre_min_val, shape = as.factor(ela_thre_min_x)),
             size = 5, color = "red", shape=1, stroke=2) +
  geom_point(aes(x = ela_thre_one_class, y = ela_thre_one_val, shape = as.factor(ela_thre_one_class)),
             size = 5, color = "red", shape=1, stroke=2) +
  scale_color_manual(values = c("Dodger Blue", "Indian Red1")) +
  labs(x='Threshold') +
  scale_x_continuous(breaks=c(ela_thre_one_class, ela_thre_min_x))

ela_thr2 = ggplot(df_thre_ela, aes(x=Percent, y=Coef_Num, color=Model)) +
  geom_line(size=1) +
  geom_point(size=2) +
  theme_light() +
  labs(title='Elasticnet, alpha 조절에 따른 피처 개수') +
  scale_color_manual(values = c("Dodger Blue", "Indian Red1")) +
      geom_point(aes(x = ela_thre_min_x, y = ela_thre_min_coef, shape = as.factor(ela_min_x)),
             size = 5, color = "red", shape=1, stroke=2) +
  geom_point(aes(x = ela_thre_one_class, y = ela_thre_one_coef, shape = as.factor(ela_one_class)),
             size = 5, color = "red", shape=1, stroke=2)
```

```
ela_thr+ theme(text = element_text(size = 10, family = "AppleMyungjo"))
```




```
ela_thr2+ theme(text = element_text(size = 10, family = "AppleMyungjo"))
```



위의 두 결과를 살펴보면 Elastic_min과 같은 경우 alpha값이 0.1인 경우 가장 성능이 좋으며, Elastic_1se의 경우 alpha값이 0.9인 경우 가장 성능이 좋다는 것을 알 수 있다.

추가적으로 변수를 선택하는 기능이 없는 randomforest에 Elastic model을 통해 선택된 변수만을 사용한다면 결과가 어떻게 될 지 궁금하여 위의 결과 중 성능이 가장 좋은 모델인 Elastic_Min_0.99에서 계수가 0이 아닌 변수만을 추출해보았다.

RandomForest 선택된 변수만으로 모델링

```
# Elastic_Min_0.99 모델 다시 구현
result = split_process(review_tfidf, 0.99, data)
df_train = result[[1]]; df_val = result[[2]]; df_test = result[[3]]
trainx = model.matrix(Y~., data=df_train)[, -1]
trainy = df_train$Y
```

```
valx = model.matrix(Y~., data=df_val)[, -1]
valy = df_val$Y
set.seed(123); cv_model = cv.glmnet(x=trainx, y=trainy, alpha=0.15, family='binomial',
                                    type.measure='class', nfolds=10)
pred = predict(cv_model, s=cv_model$lambda.min, newx=valx, type='class')
cm = confusionMatrix(factor(pred), valy, positive='negative')
acc = cm$overall['Accuracy']

# best model에서 사용된 변수만 추출
coeffs <- coef(cv_model, s = cv_model$lambda.min)
selected_vars = row.names(coeffs)[coeffs[, 1][-1] != 0]

# 해당 데이터로 구성
result = split_process(review_tfidf, 0.995, data)
df_train_ori = result[[1]]; df_val_ori = result[[2]]
df_train = df_train_ori[, selected_vars, drop = FALSE]
df_train$Y = df_train_ori$Y
df_val = df_val_ori[, selected_vars, drop = FALSE]
df_val$Y = df_val_ori$Y

# 랜덤포레스트 돌려보기
set.seed(123); sel_rf = randomForest(Y~., df_train)
sr_pred_train = predict(sel_rf, newdata=df_val, type='class')
sr_acc_rf_train = confusionMatrix(sr_pred_train, df_val$Y, positive='negative')$overall['Accuracy']
sr_acc_rf_train
```

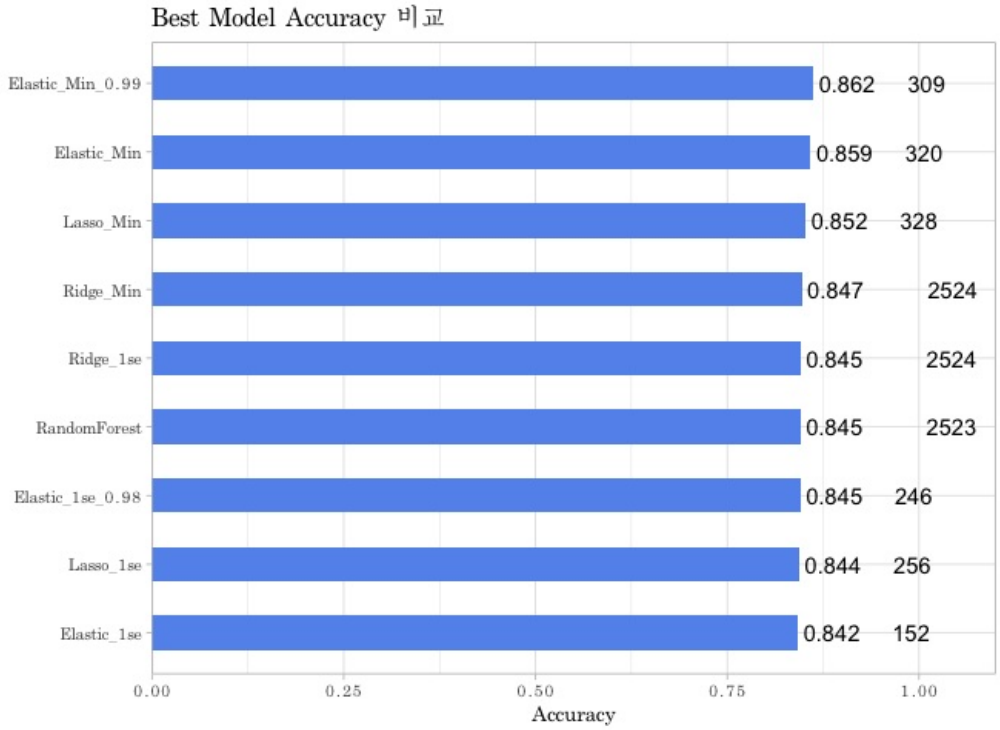
```
##  Accuracy
## 0.7154742
```

Elastic net을 통해 선택된 변수만을 사용하여 랜덤포레스트를 수행한 결과 정확도가 아주 떨어진 것을 확인하였다. 이는 규제를 통해 선택된 값이 조정되는 Elastic net과 다르게, 값의 변경 없이 선택된 변수만을 사용하여 결과가 낮지 않을까 추측하였다. 혹은 변수 수 자체가 너무 적어 랜덤포레스트 모델에서 underfitting 된 것이 아닐까 추측할 수 있다.
(위의 결과는 Best model 비교에서 제외하였다.)

추가적으로 과제 수행시 성능이 항상 높았던 svm모델을 돌려보고 싶었으나 모델이 돌아가지 않아 실패하였다. 또한 tree계열의 모델은 bagging보다 성능이 좋은 randomforest만을 사용하였으며 전처리 과정에서 가장 성능이 높았던 모델을 모델 비교 후보에 넣었다.
(xgboost같은 다른 트리계열 모델을 시도해보았으나 실패하였다,,,)

4.3 Train Best Model 선택

```
result_models = data.frame(Best_Model=c('RandomForest', 'Ridge_Min', 'Ridge_1se',
                                        'Lasso_Min', 'Lasso_1se', 'Elastic_Min',
                                        'Elastic_1se', 'Elastic_Min_0.99', 'Elastic_1se_0.98'),
                           Accuracy=c(tf_val, r_min_val, r_one_val,
                                      l_min_val, l_one_val, ela_min_val,
                                      ela_one_val, ela_thre_min_val, ela_thre_one_val),
                           Coef_Num=c(tf_coef, r_min_coef, r_one_coef,
                                      l_min_coef, l_one_coef, ela_min_coef,
                                      ela_one_coef, ela_thre_min_coef, ela_thre_one_coef))

ggplot(result_models, aes(x=reorder(Best_Model, Accuracy), y=Accuracy)) +
  geom_bar(stat='identity', width=0.5, fill='cornflowerblue') +
  theme_light() +
  coord_flip() +
  labs(x='', title = 'Best Model Accuracy 비교') +
  geom_text(aes(label=round(Accuracy, 3)), hjust=-0.1) +  # 해당 모델의 accuracy값
  geom_text(aes(label=round(Coef_Num, 3)), hjust=-2.5) +  # 모델에 사용된 변수 개수
  scale_y_continuous(expand=c(0,0), limits=c(0, 1.1)) +
  theme(text = element_text(size = 10, family = "AppleMyungjo"))
```



위의 그래프에서 막대 끝 숫자 중 왼쪽은 정확도를 오른쪽은 해당 모델에서 사용된 변수의 개수를 의미한다. 위의 결과를 통해 Elastic_Min에서 가장 높은 성능을 보였던 0.99퍼센트에서 alpha값을 조절하여 구한 Elastic_Min_0.99가 가장 높은 정확도를 갖음을 확인하였다. 따라서 해당 모델을 Best model로 선택하였다.

추가적으로 변수의 개수가 300개 정도인 경우가 대체적으로 성능이 좋고 2000개가 넘는 경우 중간, 300개 아래인 경우 낮은 것을 확인할 수 있다. 따라서 과적합을 피하기 위해 변수를 줄이는 것이 좋지만 300개 아래는 너무 적다고 판단할 수 있다. 하지만 이 역시 해당 데이터를 통해 판단한 것이므로 새로운 데이터를 통해 살펴본다면 모델의 결과가 바뀔수도 있을 것이라 예상된다.

4.4 Test set으로 결과 비교

4.3 과정을 통해 Best model은 Elastic_Min_0.99로 선택하였지만 다른 모델들의 결과도 함께 비교하고자 한다. 따라서 해당 모델들을 validation을 나누기 전 train set을 사용하여 다시 학습한 후 test를 진행하였다.

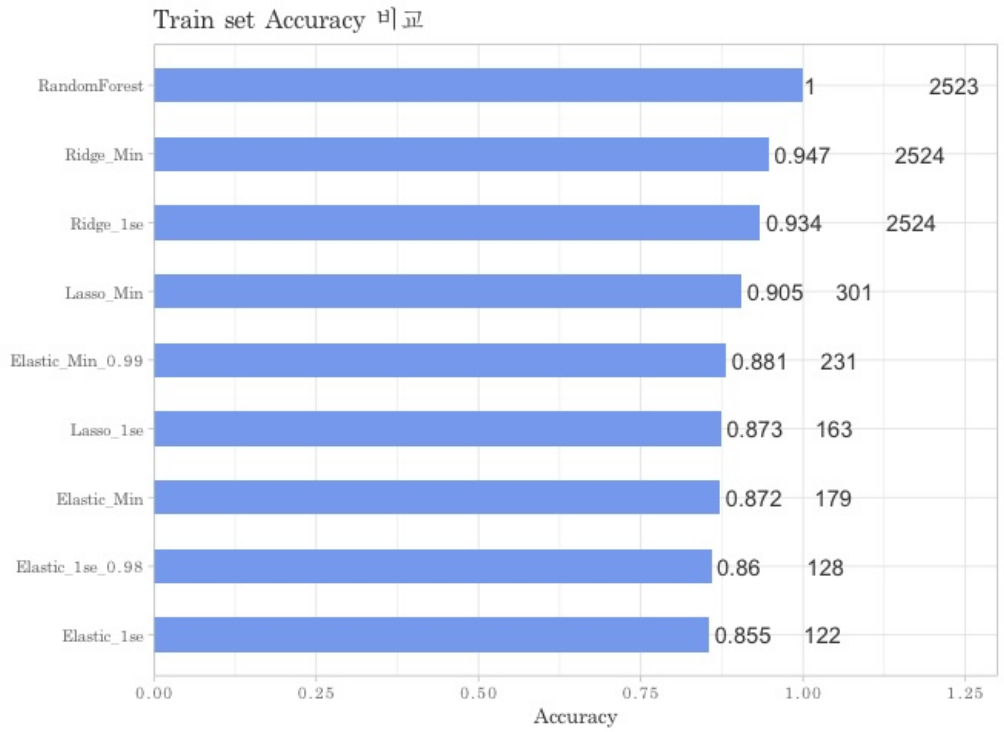
```
# validation set을 나누기 전 모든 train set을 사용하기 위한 함수
split_new_process = function(x, y, z){
  dtm = removeSparseTerms(x, y)
  df_m = data.frame(as.matrix(dtm))
  df_m$Y = z$sentiment
  df_total = df_m[1:3000, ]
  df_test = df_m[3001:10000, ]
  return (list(df_total, df_test))
}

# RandomForest
result = split_new_process(review_tfidf, 0.995, data)
df_total = result[[1]]; df_test = result[[2]]
set.seed(123); final_rf = randomForest(Y~., df_total)
pred_train = predict(final_rf, newdata=df_total, type='class')
acc_rf_train = confusionMatrix(pred_train, df_total$Y, positive='negative')$overall['Accuracy']
pred_test = predict(final_rf, newdata=df_test, type='class')
acc_rf_test = confusionMatrix(pred_test, df_test$Y, positive='negative')$overall['Accuracy']


# Ridge_Min
result = split_new_process(review_tfidf, 0.995, data)
df_total = result[[1]]; df_test = result[[2]]
trainx = model.matrix(Y~., data=df_total)[, -1]
trainy = df_total$Y
valx = model.matrix(Y~., data=df_test)[, -1]
```

```
valy = df_test$Y
acc_rm_train = min_selection_model(trainx, trainy, trainx, trainy, 0)
acc_rm_test = min_selection_model(trainx, trainy, valx, valy, 0)


# Ridge_1se
result = split_new_process(review_tfidf, 0.995, data)
df_total = result[[1]]; df_test = result[[2]]
trainx = model.matrix(Y~., data=df_total)[, -1]
trainy = df_total$Y
valx = model.matrix(Y~., data=df_test)[, -1]
valy = df_test$Y
acc_ro_train = one_selection_model(trainx, trainy, trainx, trainy, 0)
acc_ro_test = one_selection_model(trainx, trainy, valx, valy, 0)


# Lasso_Min
result = split_new_process(review_tfidf, 0.99, data)
df_total = result[[1]]; df_test = result[[2]]
trainx = model.matrix(Y~., data=df_total)[, -1]
trainy = df_total$Y
valx = model.matrix(Y~., data=df_test)[, -1]
valy = df_test$Y
acc_lm_train = min_selection_model(trainx, trainy, trainx, trainy, 1)
acc_lm_test = min_selection_model(trainx, trainy, valx, valy, 1)


# Lasso_1se
result = split_new_process(review_tfidf, 0.985, data)
df_total = result[[1]]; df_test = result[[2]]
trainx = model.matrix(Y~., data=df_total)[, -1]
trainy = df_total$Y
valx = model.matrix(Y~., data=df_test)[, -1]
valy = df_test$Y
acc_lo_train = one_selection_model(trainx, trainy, trainx, trainy, 1)
acc_lo_test = one_selection_model(trainx, trainy, valx, valy, 1)


# Elastic_Min
result = split_new_process(review_tfidf, 0.99, data)
df_total = result[[1]]; df_test = result[[2]]
trainx = model.matrix(Y~., data=df_total)[, -1]
trainy = df_total$Y
valx = model.matrix(Y~., data=df_test)[, -1]
valy = df_test$Y
acc_em_train = one_selection_model(trainx, trainy, trainx, trainy, 0.5)
acc_em_test = one_selection_model(trainx, trainy, valx, valy, 0.5)


# Elastic_1se
result = split_new_process(review_tfidf, 0.98, data)
df_total = result[[1]]; df_test = result[[2]]
trainx = model.matrix(Y~., data=df_total)[, -1]
trainy = df_total$Y
valx = model.matrix(Y~., data=df_test)[, -1]
valy = df_test$Y
acc_eo_train = one_selection_model(trainx, trainy, trainx, trainy, 0.5)
acc_eo_test = one_selection_model(trainx, trainy, valx, valy, 0.5)


# Elastic_Min_0.99
result = split_new_process(review_tfidf, 0.99, data)
df_total = result[[1]]; df_test = result[[2]]
trainx = model.matrix(Y~., data=df_total)[, -1]
trainy = df_total$Y
valx = model.matrix(Y~., data=df_test)[, -1]
valy = df_test$Y
```

```r
acc_eo99_train = one_selection_model(trainx, trainy, trainx, trainy, 0.15)
acc_eo99_test = one_selection_model(trainx, trainy, valx, valy, 0.15)


# Elastic_1se_0.98
result = split_new_process(review_tfidf, 0.98, data)
df_total = result[[1]]; df_test = result[[2]]
trainx = model.matrix(Y~., data=df_total)[, -1]
trainy = df_total$Y
valx = model.matrix(Y~., data=df_test)[, -1]
valy = df_test$Y
acc_eo98_train = one_selection_model(trainx, trainy, trainx, trainy, 0.9)
acc_eo98_test = one_selection_model(trainx, trainy, valx, valy, 0.9)

# Train set accuracy 결과 저장
df_result_train = data.frame(Model=c('RandomForest', 'Ridge_Min', 'Ridge_1se',
                                     'Lasso_Min', 'Lasso_1se', 'Elastic_Min',
                                     'Elastic_1se', 'Elastic_Min_0.99', 'Elastic_1se_0.98'),
                             Accuracy=c(acc_rf_train, acc_rm_train[[1]], acc_ro_train[[1]],
                                        acc_lm_train[[1]], acc_lo_train[[1]], acc_em_train[[1]],
                                        acc_eo_train[[1]], acc_eo99_train[[1]], acc_eo98_train[[1]]),
                             Coef_Num=c(tf_coef, acc_rm_train[[2]], acc_ro_train[[2]],
                                        acc_lm_train[[2]], acc_lo_train[[2]], acc_em_train[[2]],
                                        acc_eo_train[[2]], acc_eo99_train[[2]], acc_eo98_train[[2]]))

df_result_train['Data'] = 'Train Set'

# Test set accuracy 결과 저장
df_result_test = data.frame(Model=c('RandomForest', 'Ridge_Min', 'Ridge_1se',
                                    'Lasso_Min', 'Lasso_1se', 'Elastic_Min',
                                    'Elastic_1se', 'Elastic_Min_0.99', 'Elastic_1se_0.98'),
                            Accuracy=c(acc_rf_test, acc_rm_test[[1]], acc_ro_test[[1]],
                                       acc_lm_test[[1]], acc_lo_test[[1]], acc_em_test[[1]],
                                       acc_eo_test[[1]], acc_eo99_test[[1]], acc_eo98_test[[1]]),
                           Coef_Num=c(tf_coef, acc_rm_test[[2]], acc_ro_test[[2]],
                                       acc_lm_test[[2]], acc_lo_test[[2]], acc_em_test[[2]],
                                       acc_eo_test[[2]], acc_eo99_test[[2]], acc_eo98_test[[2]]))
df_result_test['Data'] = 'Test Set'


# Train set 결과 비교
p_train = ggplot(df_result_train, aes(x=reorder(Model, Accuracy), y=Accuracy)) +
  geom_bar(stat='identity', width=0.5, fill='cornflowerblue') +
  theme_light() +
  coord_flip() +
  labs(x='', title = 'Train set Accuracy 비교') +
  geom_text(aes(label=round(Accuracy, 3)), hjust=-0.1) +
  geom_text(aes(label=round(Coef_Num, 3)), hjust=-2.5) +
  scale_y_continuous(expand=c(0,0), limits=c(0, 1.3))

# Test set 결과 비교
p_test = ggplot(df_result_test, aes(x=reorder(Model, Accuracy), y=Accuracy)) +
  geom_bar(stat='identity', width=0.5, fill='cornflowerblue') +
  theme_light() +
  coord_flip() +
  labs(x='', title = 'Test set Accuracy 비교') +
  geom_text(aes(label=round(Accuracy, 3)), hjust=-0.1) +
  geom_text(aes(label=round(Coef_Num, 3)), hjust=-2.5) +
  scale_y_continuous(expand=c(0,0), limits=c(0, 1.3))

df_total_result = rbind(df_result_train, df_result_test)
over = ggplot(df_total_result, aes(x=Model, y=Accuracy, color=Model)) +
  geom_line(size=1) +
  geom_point(size=2) +
  theme_light() +
  labs(title='Train, Test Accuracy 비교') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Train set Accuracy

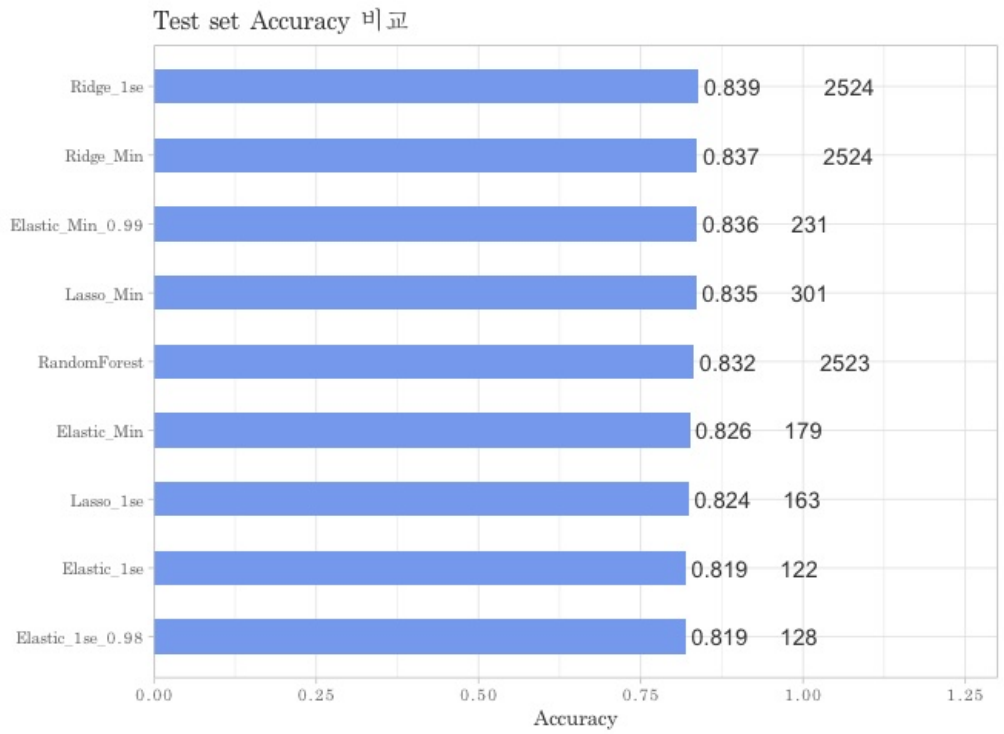
```
p_train+ theme(text = element_text(size = 10, family = "AppleMyungjo"))
```



먼저 Train set의 정확도를 살펴봤을때 Randomforest가 가장 높은 것을 확인할 수 있다. 하지만 정확도가 1로 과적합 되었다는 것을 알 수 있다. 또한 Train Best model에서 변수의 개수가 300개 정도였을때 비교적 높았던 것과 다르게 위의 결과에서는 변수의 개수가 많을수록 정확도가 높은 것을 확인할 수 있다. 이는 해당 변수가 많을수록 과적합 될 수 있다는 것을 의미한다.

Test set Accuracy

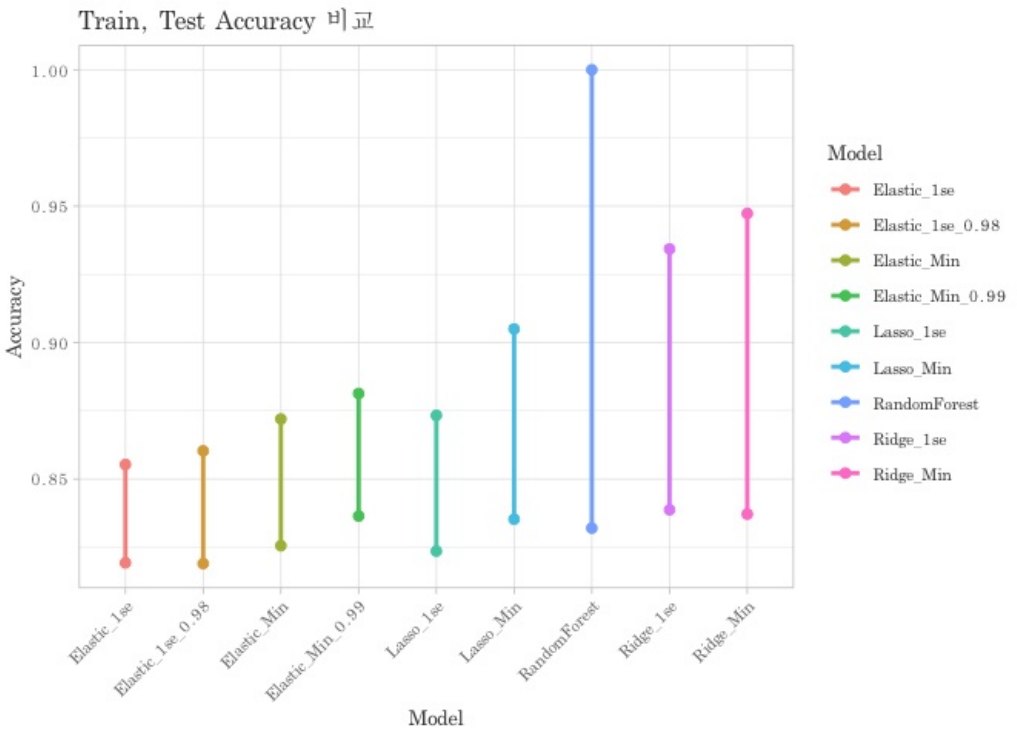
```
p_test+ theme(text = element_text(size = 10, family = "AppleMyungjo"))
```



다음으로 Test set을 통해 결과를 비교해보면 Best model로 선택했던 Elastic_Min_0.99가 3등 인것을 확인할 수 있다. 1등과 2등은 Ridge 모델로 변수의 개수가 많으면 과적합 될 것이라 예상했던 것과 다르게 성능이 가장 좋았다. 하지만 best model로 선택했던 모델이 200대 정도의 변수로 예측한 성능과 0.03 정도 차이로 크지 않다는 것을 알 수 있다. 따라서 정확도로만 본다면 Ridge모델이 더 좋았지만 효율면에서는 best model로 선택했던 것이 더 좋다고 판단된다. 또한 변수를 적게 사용하는 모델을 예상과 동일하게 모두 성능이 낮은 것을 확인하였다. 추가적으로 one-standard-rule을 적용한 모델이 대체로 낮았던 것과 다르게 Test set에서는 1등을 차지한 것을 통해 변수를 과하게 줄이지 않는다면 단순한 모델이 일반화 성능이 더 좋다는 것을 확인할 수 있었다.

Overfitting 확인

```
over+ theme(text = element_text(size = 10, family = "AppleMyungjo"))
```



다음으로 Train accuracy와 Test accuracy를 비교하기 위해 그림을 그려보았다. 위의 결과를 통해 RandomFoest모델이 과적합이 되었다는 것을 알 수 있다. 추가적으로 Ridege모델 역시 test set에 대한 정확도가 높긴 하였지만 train이 월등하게 높은 것을 보아 Train set에 다른 모델보다 더 과적합 되었다는 것을 알 수 있다. 또한 변수의 개수가 작을수록 Train set과 Test set의 성능 차이가 적다는 것 역시 알 수 있다.