Índice

1. Introducción

Se realizo una competencia de Kaggle la cual tenía como objetivo realizar una predicción binaria a partir de un set de datos el cual contenía un total de 250 filas para realizar el entrenamiento del modelo y 300 variables, la mayor dificultad de esta competencia era evitar el sobreajuste.

A continuación se mostraran las diferentes dificultades que se tuvieron para resolver el problema y los resultados de los diferentes algoritmos utilizados y el porque de cada uno, tomando en cuenta que el sobre ajuste es un problema el cual siempre se tendrá que tomar en cuenta al realizar predicciones a partir de un set de datos.

2. Competencia (Don't Overfit! II)

En esta competencia de Kaggle se tenía un problema de clasificación binaria. Se Brindaba la siguiente información:

- Datastet de 250 filas.
- 300 variables nombradas de la x0 hasta la x299.
- Target que puede tener valores 0 y 1.

El propósito era lograr predecir 19,750 nuevos targets que se encuentran en el dataset de prueba, aparte de predecir el target se debía evitar el sobre ajuste de los datos. Es importante mencionar que todos los datos dados eran numéricos, y no existían datos nulos o datos NA.

3. Hallazgos

- Varios algoritmos de selección de variables: Se encontró que se contaba con algoritmos que ya hacían el trabajo de decidir que variables eran significantes o no pero no se sabia cual era el mas eficiente por lo tanto se debió probar con cada uno de ellos y realizar comparaciones.
- Varios algoritmos de predicción: se encontraron muchos algoritmos para clasificación binaria pero el problema estaba ahora en cual seria el mejor prediciendo y evitando el sobre ajuste.
- Regularización LASSO, Ridge, Elastic Net: entre todos los algoritmos de regularización cual seria el mas eficiente para el problema presentado, una ventaja es que se encontró la librería **glmnet** la cual proporciona los 3 tipos de regularización.

4. Dificultades

- No sabíamos que estábamos prediciendo: este fue uno de los mayores retos durante la competencia ya que no teníamos ninguna idea de que variables podíamos tomar en cuenta dado que el valor a predecir era un target el cual era 1 o 0 y los nombres de las variables eran del x0 al x299 por lo tanto no se tenía mucha información de lo que se quería predecir.
- Se tenían el pensamiento que muy pocos datos eran proporcionados para predecir una gran cantidad de datos: proporcionaron un total de 250 filas y la cantidad de datos a predecir era de 19,750 por lo tanto la diferencia en porcentaje de los datos para entrenamiento era del 1.27 % en comparación de los datos a predecir.
- Escoger entre 300 variables continuas: las variables eran con valores con muchos decimales, negativos, positivos por lo tanto tuvimos que seleccionar solamente las variables que si eran significante para el modelo.
- Más variables que muestras.

5. Algoritmo de Solución

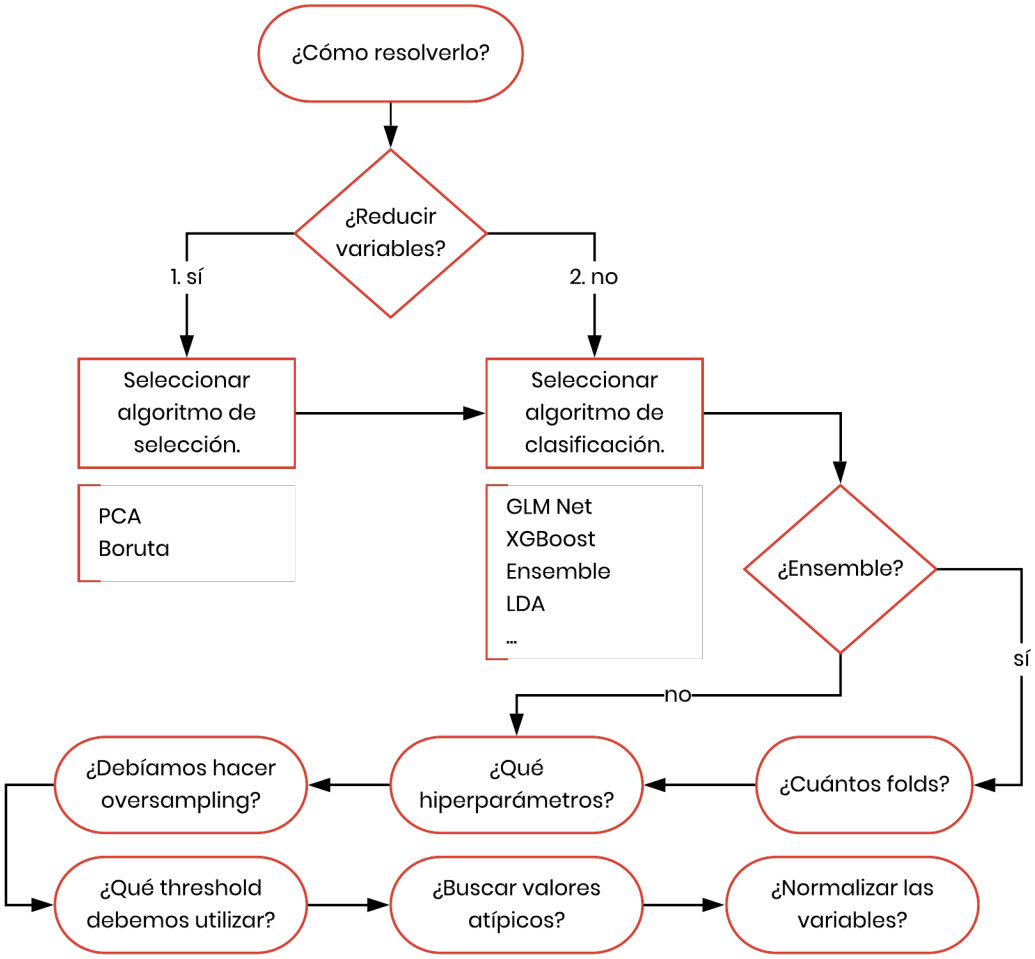


Figura 1: Diagrama de flujo de la resolución de problema.

1. Como se menciono anteriormente no se sabía que era lo que se debía predecir por lo tanto la primera idea que se tuvo fue la eliminación de variables que no eran significantes para el modelo.

2. Dado que se empezó con la selección de variables, luego de investigar acerca de algoritmos, cada uno de ellos fueron utilizados para cuales eran los resultados se obtenidos y solamente dejando las variables que eran significantes según los algoritmos el mejor resultado se obtuvo utilizando Boruta por lo cual se decidió continuar con dicho algoritmo.

3. Luego se indago acerca de algoritmos para solucionar el problema y se probaron una gran cantidad de algoritmos y se seleccionaron los algoritmos que dieron un mejor resultado los cuales fueron: regresión logística regularizada y conjuntos.

4. Luego de consideran cuales de los algoritmos ya seleccionados era el mejor se llegó a la conclusión que era el de conjuntos teniendo un resultado mejor que el de regresión logística regularizada.

5. Al ya definido que el algoritmo de conjuntos sería el utilizado, se tuvieron las siguientes consideraciones.

   a) **¿Cuántos folds se utilizarían?**
   Un fold en el algoritmo de conjuntos significa que los datos serán divididos en la cantidad de folds seleccionada, por lo tanto no se sabía realmente en cuantos folds el algoritmo seria mejor y se fue probando poco a poco y se utilizo la cantidad de folds que generaban una mejor predicción.

   b) **¿Qué hiperparámetros y threshold utilizar?**
   De igual forma se fueron probando con distintos valores y se utilizaban los que generaban una mejora en la predicción.

   c) **¿Debíamos hacer oversampling o no?**
   Dado que los datos para entrenar eran muy pocos, se tuvo la idea de generar mas datos utilizando algoritmos que en base a los que ya se tienen pueden generar mas datos, por ejemplo pasar de las 250 filas a 350 filas o más, pero se decidió no hacerlo dado que de por si los datos ya generaban sobre ajuste, tener mas datos obtenidos a partir de los mismos datos aumentaría el sobre ajuste.

   d) **¿Eliminar valores atípicos?**
   Se pensó en eliminar valores que eran muy grandes para ver si la predicción mejoraba, pero luego se decidió mejor no hacerlo dado que de por si la cantidad de datos para entrenar el eliminar valores atípicos solamente la reduciría.

   e) **¿Necesidad de normalizar valores?**
   Los valores fueron normalizados tanto para la regresión logística regularizada y conjuntos, pero se observó que la regresión logística regularizada si mejoraba la predicción al normalizar los valores a diferencia de los conjuntos el resultado de las predicciones se veían afectados de una manera negativa por lo tanto en dicho algoritmo se decidió no normalizar.

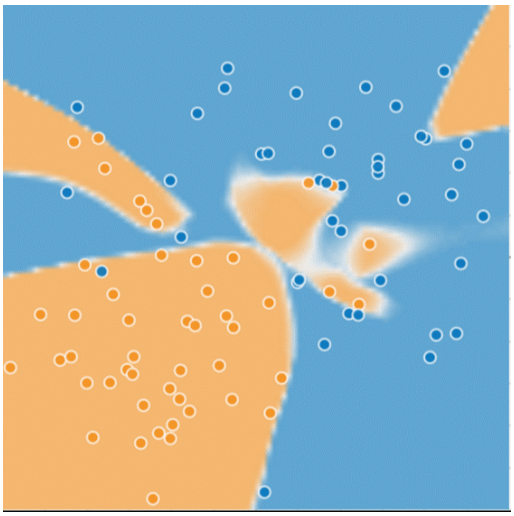
6. Marco Teórico

6.1. ¿Qué es overfit?

Es un problema que se da en los modelos de machine learning, el cual consiste en que el modelo se ajusta muy bien a los datos de entrenamiento y esto provoca que al predecir con nuevos datos, este no predice correctamente.

En el siguiente ejemplo se tiene una cierta cantidad de datos que tienen dos clases, la clase de color anaranjada y la clase de color azul. La segunda imagen muestra el resultado de nuestro modelo, las regiones de color azul predicen que el dato es de color azul, y las regiones anaranjadas predicen que el dato es anaranjado, y en ese caso se puede observar que el modelo funciona casi con un 100 % de exactitud.

(a) Dataset de entrenamiento.



(b) Modelo complejo.

Los datos mostrados en la primera imagen (a) son solo un subconjunto de todos los datos que se tiene para entrenar el modelo, en la siguiente imagen (b) podemos observar el resto de los datos, sobre la imagen que tiene el modelo, en la esquina inferior derecha de la imagen se puede observar que hay varios puntos anaranjados en una región que el modelo predice como azul, y también podemos ver que a la izquierda la mayoría de los datos son anaranjados, pero el modelo predijo una pequeña región de datos azules del lado izquierdo.

En la figura 4 se puede observar que un modelo mas simple compuesto de una línea recta funcionaría mejor en este caso, a diferencia del modelo complejo que se mostró en las imágenes anteriores.

Existen diferentes métodos para reducir el overfit, el método más sencillo es cambiar el orden de los datos a un orden aleatorio y luego dividir los datos diferentes conjuntos de validación y test. Otro método mas efectivo pero que también puede causar problemas es la regularización. Así como existe overfit, también existe el contrario que es llamado underfit, el cual es un problema donde el modelo no con los datos de entrenamiento ni con los demás datos. Y se puede dar cuando se abusa del uso de la regularización.

En el primer ejemplo se muestra un modelo de predicción de imágenes de perros, el underfit se da debido a que no se utilizaron suficientes datos de entrenamiento.

En el segundo ejemplo se utilizaron mas datos de entrenamiento sin embargo todas las imágenes utilizados fueron de perros de color marrón, entonces el modelo se sobre ajustó a los perros de ese color, y no funciona para predecir perros de otro color.

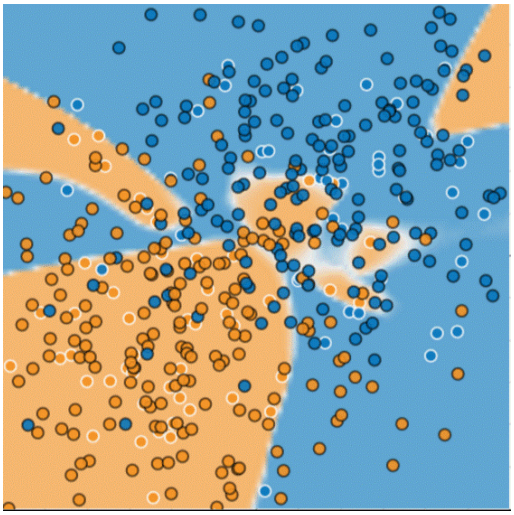


Figura 3: Datos de la población.

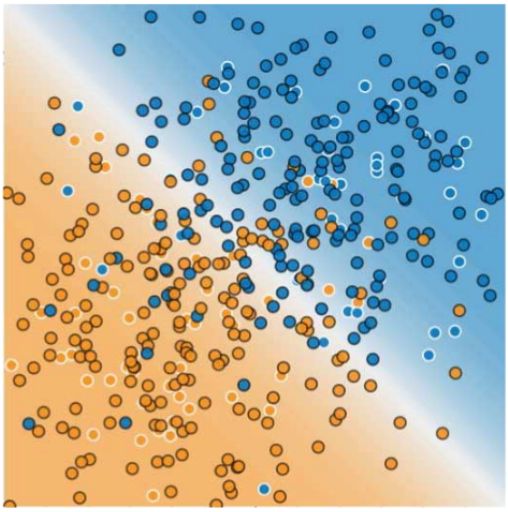


Figura 4: Datos de la población.

6.2. ¿Qué es regularización?

La regularización en machine learning es un método que se utiliza para evitar el problema de overfit. El objetivo que tiene los métodos de regularización es disminuir la complejidad de un modelo o también se puede decir que quiere hacer un modelo más simple. La complejidad del modelo se puede calcular en función de la cantidad de variables que utiliza el sistema o en cantidad de la magnitud de los pesos.

Al entrenar un modelo de machine el principal objetivo es minimizar el error del modelo respecto de los datos de entrenamiento, pero la regularización también tiene el objetivo de minimizar la complejidad del modelo de la siguiente forma:

Figura 5: Underfitting.

Figura 6: Overfitting.

$$Minimize(Loss(DataModel) + \lambda Complexity(Model))$$

En esta expresión podemos observar un parámetro lambda multiplicado a la complejidad del modelo, este parámetro nos indica que tanta importancia le queremos dar a la minimización de la complejidad, si lambda es 0 significa que no queremos minimizar la complejidad del modelo y si es 1 significa que si le estamos dando mucha importancia a la complejidad del modelo.

Hay dos tipos principales que son regularización L1 y L2. La regularización L1 toma la complejidad en función del valor absoluto de los pesos, el mas conocido es la regularización LASSO. La regularización L2 toma la complejidad en función de el cuadrado de los pesos, el mas conocido es la regularización Ridge.

En la siguiente imagen podemos observar una ecuación de regularización donde podemos observar que se trata de una minimización, el primer termino de la suma dentro de la suma-

toria es el error del modelo respecto a los datos y el segundo termino es la complejidad del modelo, también podemos observar el parámetro lambda. La grafica azul es una regresión sin regularización y podemos observar que tiene muchos cambios grandes debido a que se quiere ajustar los mas posible a los datos de entrenamiento, la grafica roja es una regresión que, si utilizo regularización, podemos observar que no hay tantos cambios muy grandes como en la gráfica azul ya que es una regresión mas general que puede dar mejores resultados con otros datos que no estaban en el entrenamiento.

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda \left[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1 \right]$$

Figura 7: Gráfica que describe el sobreajuste.

6.3. Regresión Logística

La regresión logística es utilizada para resolver problemas de clasificación binaria y es el método de menor complejidad, por lo que con esta se logra un menor overfit que con otros metodos.

La regresión logística tiene la siguiente forma:

$$p_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k)}}$$

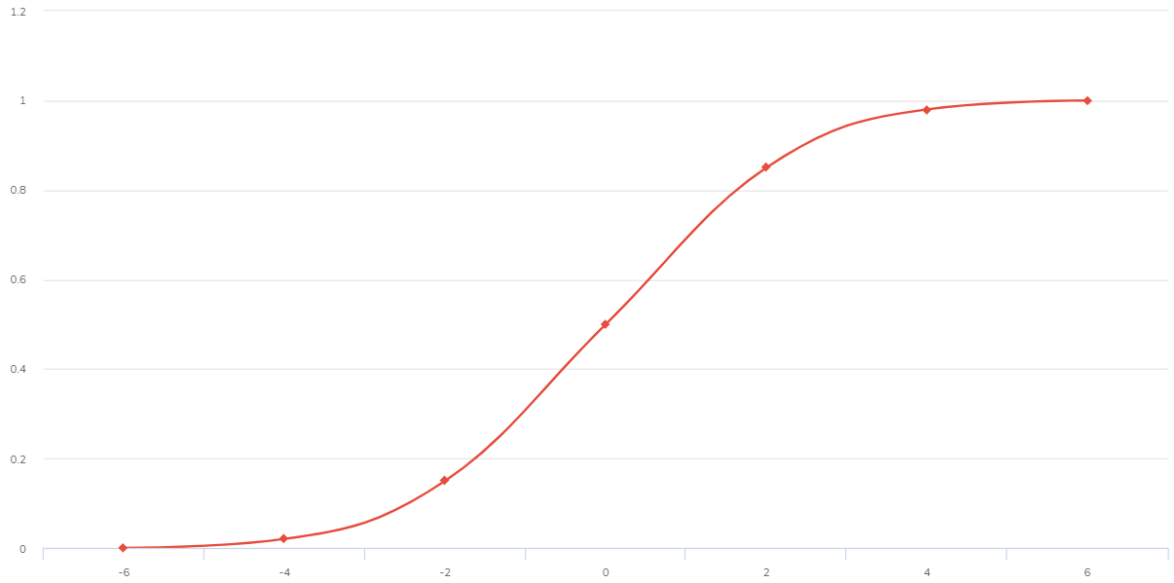


Figura 8: Regresión logística.

La función siempre devuelve valores entre 0 y 1 que pueden ser representados como una probabilidad, como podemos observar en la ecuación existe la ecuación de una regresión lineal en el que se encuentran las variables de nuestro dataset y sus respectivos pesos.

7. Cross validation (CV)

Cross validation es una técnica utilizada para entrenar modelos de machine learning cuando se tiene una cantidad limitada de datos de entrenamiento. Esta técnica se basa en realizar diferentes subconjuntos del set de datos entero para entrenar el modelo. Para realizar cross validation se siguen los siguientes pasos:

1. Reordenar todos los datos de forma aleatoria.
2. Dividir el dataset en k grupos.
3. Para cada grupo hacer lo siguiente:

a) Utilizar el grupo como dataset de prueba.

b) Utilizar los grupos restantes para el entrenamiento.

c) Realizar el modelo que se ajuste a los datos y evaluarlo con el grupo de dataset.

d) Guardar los resultados del entrenamiento.

4. Resumir la habilidad del modelo usando los resultados de entrenamiento de cada grupo.

El valor k en este procedimiento es la cantidad de grupos en los que se divide el dataset, y es determinado por el usuario.

Figura 9: Cross validation con $k = 4$.

8. Algoritmos Utilizados para solucionar el problema

8.1. Regresión logística regularizada

El primer paso en este modelo fue importar las librerías que utilizamos para las funciones de entrenar el modelo, y leer los datasets de entrenamiento y de test en R.

```
1
2 library(dplyr)
3 library(caret)
4
5 dataset <- read.csv("train.csv")
6 dataset.test <- read.csv("test.csv")
7
```

Luego de esto preparamos los datos aplicándole una escala con el método de mean scale:

$$\frac{x - \bar{x}}{\sigma}$$

Y convirtiendo la variable del target, a un valor de factor porque tenemos que predecir entre dos clases.

```
1
2  mean_tr <- apply(dataset[,-(1:2)], 2, mean)
3  sd_tr <- apply(dataset[,-(1:2)], 2, sd)
4
5  X_train <- scale(
6          dataset[,-(1:2)],
7          center=mean_tr,
8          scale=sd_tr
9      ) %>% data.frame
10
11 X_test <- scale(
12         dataset.test[,-1],
13         center=mean_tr,
14         scale=sd_tr
15     ) %>% data.frame
16
17 Y_train <- factor(
18         dataset$target,
19         levels=c(1,0),
20         labels=c("Yes", "No")
21     )
22
```

Luego de esto ya podemos realizar el entrenamiento de nuestro modelo, antes de entrenarlo le tenemos que especificar a la función que queremos utilizar el método de cross validation, esto lo hacemos con la función trainControl() el cual recibe el método que se quiere utilizar y un numero que es el parámetro k que especifica los grupos que utilizara el algoritmo. Luego utilizamos la función train para entrenar el modelo con el método de glmnet.

```
1
2  my_control <- trainControl(
3          methon="cv",
4          number=5,
5          verboseIter=T,
6          summaryFunction=twoClassSummary,
7          classProbs = TRUE
8      )
9
10 mod_elasticnet <- train(
11         x=X_train,
12         y=Y_train,
13         method="glmnet",
14         trControl=my_control,
```

```
        tuneLength=10,
        family = "binomial",
        metric="ROC"
    )

```

El método glmnet, entrena al modelo utilizando una regresión logisitica y puede seleccionar un método de regularización entre los cuales están LASSO, Ridge, y Elastic Net. Al realizar nuestro modelo obtuvimos los siguientes valores:

	α	λ
Valor	1	0.0672071

Cuadro 1: Valores obtenidos por CV.

El valor de α como 1 significa que glmnet utilizo regularización LASSO y tambien nos da el valor del parámetro λ.

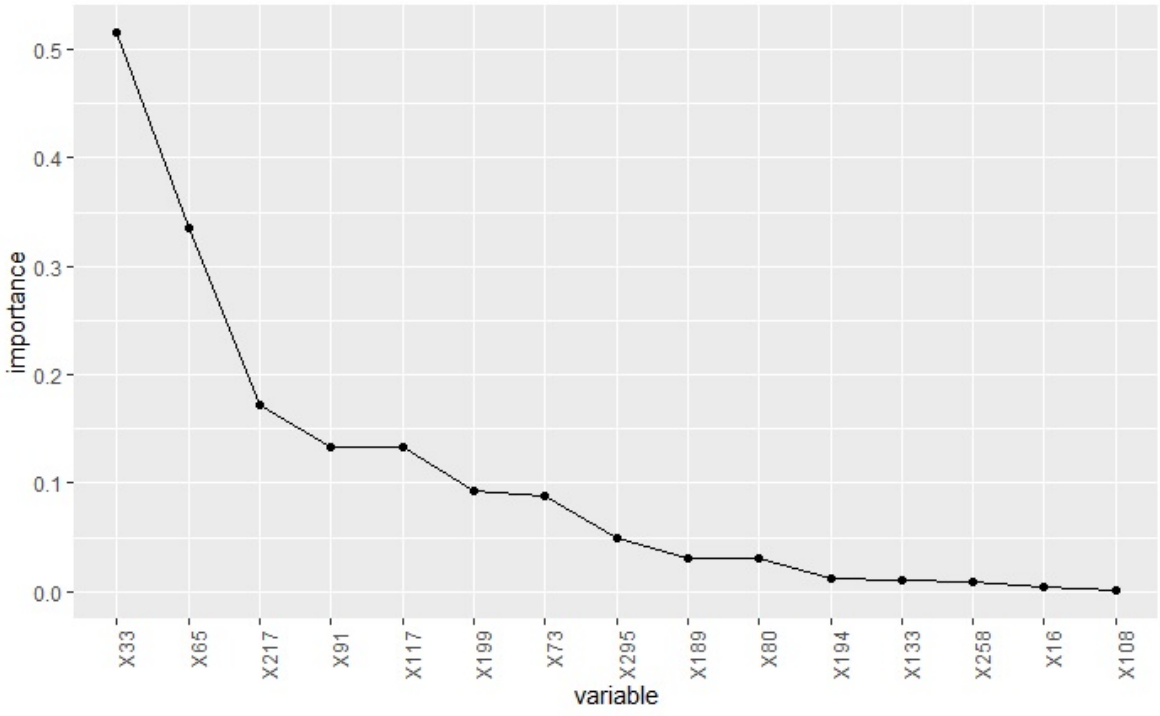


Figura 10: Variables seleccionadas.

También nos da una grafica donde nos indica la importancia que tiene las variables en nuestro modelo. Como se puede observar en la grafica la variable mas importante es X33 y a partir de la variable X258, la importancia de las variables es casi 0.

Por último, realizamos las predicciones utilizando un threshold de 0.8 en el que los mayores de este valor son de la clase 1 y el resto de la clase 0 y obtuvimos los siguientes resultados.

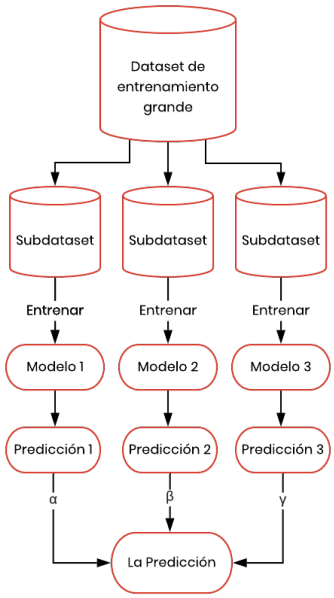
```
1
2 pred.elasticnet <- predict(mod_elasticnet, X_test, type="prob")
3 predicted.classes <- ifelse(pred.elasticnet$Yes > 0.8, 1, 0)
4
5 salida <- data.frame(id = dataset.test$id, target = predicted.
     classes)
6
7 write.csv(salida, file="KaggleOut.csv", row.names=FALSE)
8
```

8.1.1. Ranking

Resultado: 0.721
Posición: 1597 de 2291
Percentil: 30

8.2. Ensembles o Conjuntos

El método de conjuntos permite combinar las predicciones de diversos estimadores de base construidos con un algoritmo de aprendizaje el cual ayuda a mejorar la generalizabilidad sobre un único estimador. Y brinda un ahorro de tiempo ya que permite generar un análisis de los modelos en vez de estar probando cada uno de los algoritmos para cada modelo.



(a) Ejemplo de conjuntos.

```
All prediction algorithm wrappers in SuperLearner:

 [1] "SL.bartMachine"      "SL.bayesglm"         "SL.biglasso"         "SL.caret"
"SL.caret.rpart"
 [6] "SL.cforest"          "SL.dbarts"           "SL.earth"            "SL.extraTrees"
"SL.gam"
[11] "SL.gbm"              "SL.glm"              "SL.glm.interaction"  "SL.glmnet"
"SL.ipredbagg"
[16] "SL.kernelKnn"        "SL.knn"              "SL.ksvm"             "SL.lda"
"SL.leekasso"
[21] "SL.lm"               "SL.loess"            "SL.logreg"           "SL.mean"
"SL.nnet"
[26] "SL.nnls"             "SL.polymars"         "SL.qda"              "SL.randomForest"
"SL.ranger"
[31] "SL.ridge"            "SL.rpart"            "SL.rpartPrune"       "SL.speedglm"
"SL.speedlm"
[36] "SL.step"             "SL.step.forward"     "SL.step.interaction" "SL.stepAIC"
"SL.svm"
[41] "SL.template"         "SL.xgboost"

All screening algorithm wrappers in SuperLearner:

[1] "All"
[1] "screen.corP"           "screen.corRank"        "screen.glmnet"
"screen.randomForest"   "screen.SIS"
[6] "screen.template"       "screen.ttest"          "write.screen.template"
```

(b) Algoritmos de SuperLearner.

El primer paso en este modelo fue importar las librerías necesarias, tomando en cuenta que algoritmos se pondrán para evaluar el conjunto.

```
1
2 library("SuperLearner")
3 library("glmnet")
4 library("kernlab")
5
```

Luego es necesario preparar los datos.

```
1 train <- read.csv("train.csv")
2 test <- read.csv("test.csv")
3
4 y <- train$target
5 x <- data.frame(train[,3:ncol(train)])
6 x.test <- data.frame(test[2:ncol(test)])
7
```

En el algoritmo de conjuntos es necesario evaluar el riesgo y la importancia que tendrá el algoritmo, algo muy importante que se tomo en cuenta al realizar la evaluación de los modelos ksv y glmnet es que también se añadió el modelo basado en la media, dado que, si nuestros modelos son mejores que el modelo basado en la media, se está realizando modelos que si predecirán datos correctos.

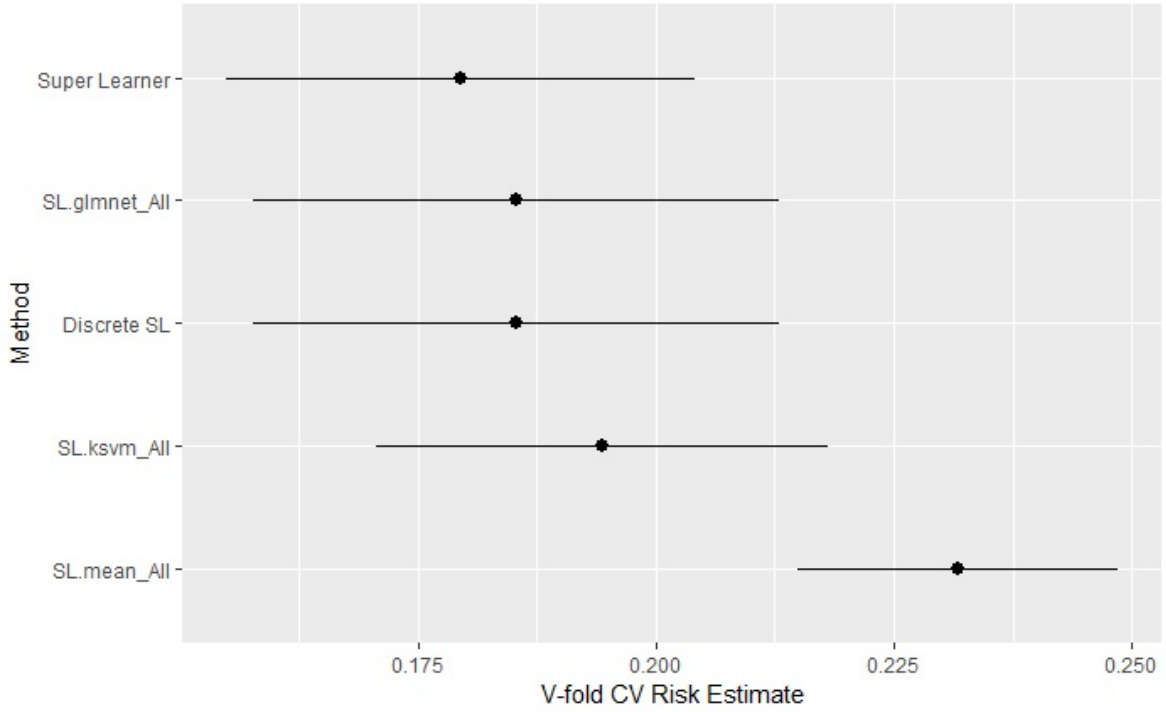


Figura 12: Rendimiento de los algoritmos.

Luego de realizar la evaluacion de los modelos y corroboran que seran buenos prediciendo, se entreno el modelo.

```
multiple.model <- mcSuperLearner(
    y,
    x,
    family = binomial(),
    cvControl = list(5),
    SL.library = list(
        "SL.ksvm",
        "SL.mena",
        "SL.glmnet"
    )
)
```

	Risk	**Coef**
SL.ksvm_All	0.1991191	0.3914801
SL.mean_All	0.2348100	0.0000000
SL.glmnet_All	0.1906389	0.6085199

Figura 13: Valores obtenidos por CV.

Luego de tener el modelo entrenado, esta listo para realizar las predicciones. A continuación se muestra el puntaje obtenido la subir el archivo a Kaggle con el modelo utilizando conjuntos.

```
predictions <- predict.SuperLearner(multiple.model, newdata = x.
   test)
binary.predictions <- ifelse(predictions$pred >= 0.65, 1, 0)

salida <- data.frame(id = test$id, target = binary.predictions)

write.csv(salida, file = "KaggleOut.csv", row.names = FALSE)
```

8.2.1. Ranking

Resultado: 0.767
Posición: 1315 de 2310
Percentil: 43

9. Conclusiones

- Los conjuntos son una gran solución con un buen rendimiento cuando existen varios algoritmos para solucionarlo, ya que proporciona un análisis de que modelo realizara una mejor predicción, evitando que estar realizando el algoritmo para cada uno de los modelos.
- Si el modelo tiene un buen rendimiento con los datos de prueba no se debe asumir que este funcionara correctamente para otros datos de prueba, dado que siempre se tendrán problemas de sobre ajuste.
- Es muy importante no abusar de la regularización porque puede conllevar a obtener underfitting en los datos.