Proyecto Final

Statistical Learning for Data Analytics - IIND 4123
Departamento de Ingeniería Industrial - Segundo Semestre de 2023
UNIVERSIDAD DE LOS ANDES
Profesor: Carlos Valencia
Estudiantes: Juan Felipe Patiño, Juan Sebastián Pardo, Santiago Zubieta

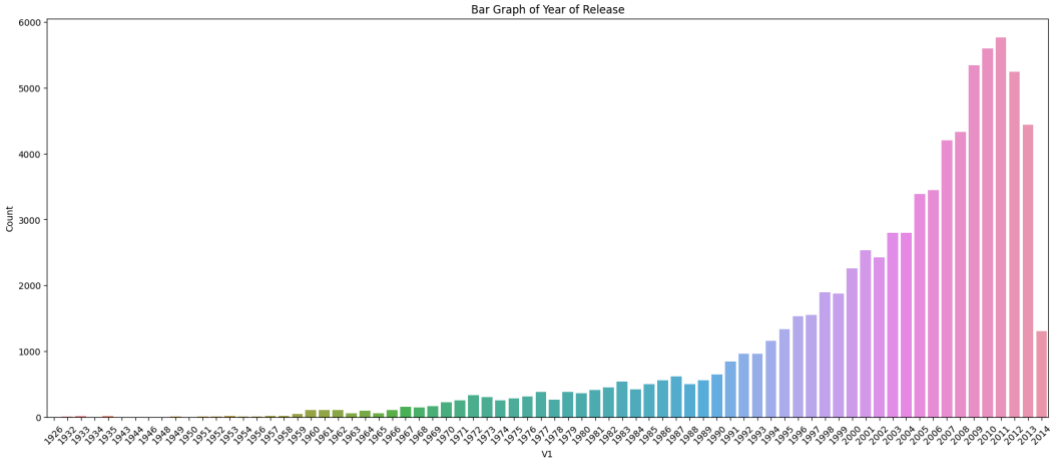
Tabla de Contenidos

1. Problema Regresión: Predicción Año Música

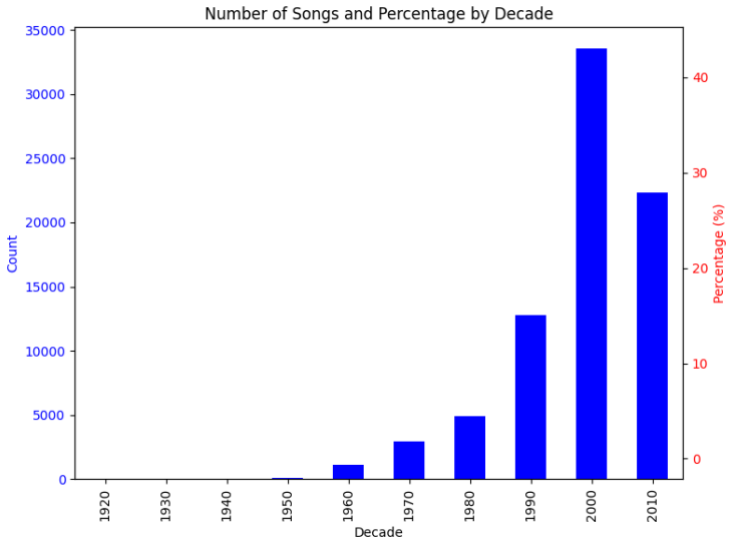
1.1. Exploración y Visualización de datos

El primer paso es entender los datos como tal. Recordemos que para la regresión deseamos predecir el año de lanzamiento de una canción a partir de datos de timbre y sus covarianzas. Esto nos genera 90 atributos en total, donde los primeros 12 con el timbre y los siguientes son las covarianzas. Esto es una versión del proyecto de *Million Song Dataset* pues son canciones nuevas, pero con algunos de los mismos atributos predictores. Primero entendamos la estructura de los datos y la distribución de nuestra variable de interés.

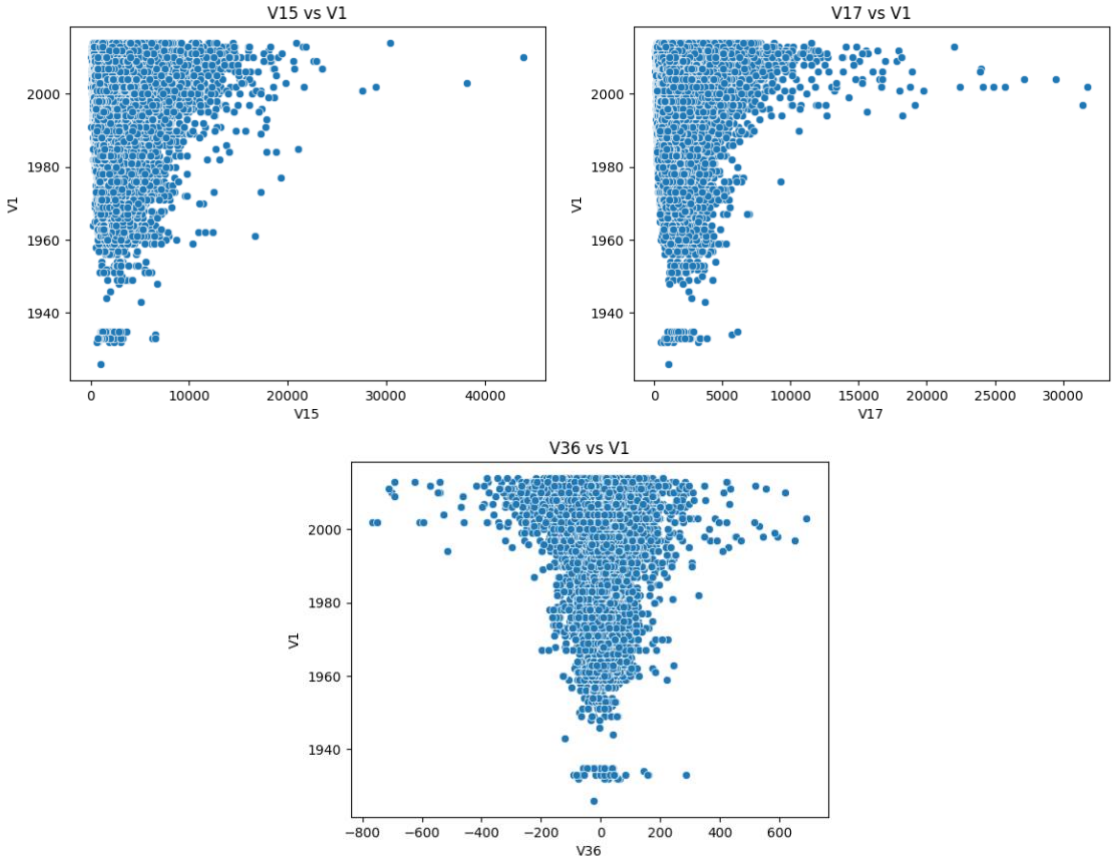
Forma datos entrenamiento	(77779, 91)
Tipos	Año: int64 V4: int64 (llena de 0) El resto: float64
Nulos	No hay



Tenemos 77,779 datos de entrenamiento, donde las variables predictoras es el int64 del año y no hay datos nulos. Es decir que es un *dataset* bastante bueno y limpio y más grande que el de clasificación. Ahora bien, podemos ver una gran distribución de años entre 1926 hasta 2014, o sea 90 años posibles para predecir, lo cual es un campo bastante amplio para el tamaño de los datos y el desbalanceo de clases, en especial al tener un par de décadas representando el un gran porcentaje del total de los datos.

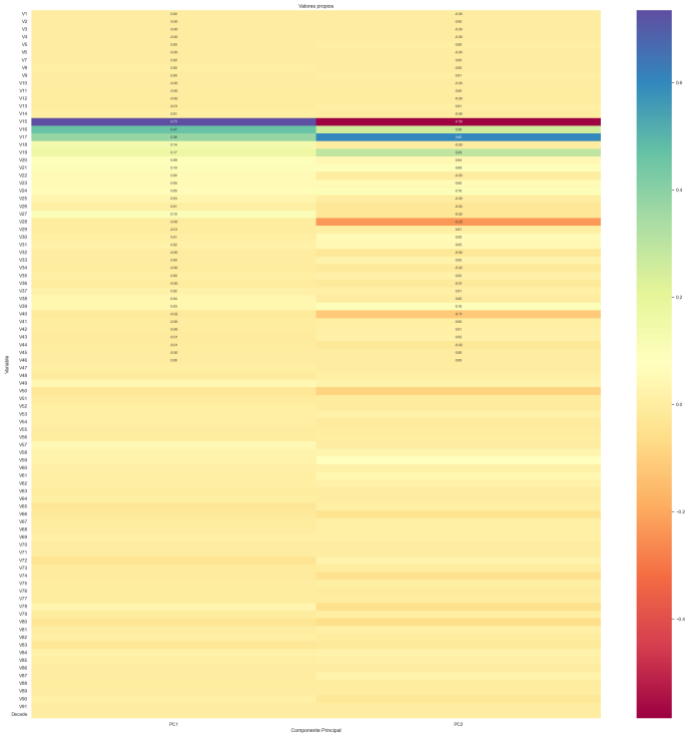


En particular los 2000 son el 40% y sumado con los 4 años del 2010 en el *dataset*, estos son casi el 70% del total de los datos. Ahora procedemos a analizar las relaciones entre la variable objetivo, 'V1, y cada variable por medio de un gráfico. En este caso por motivos de espacio solo incluiremos las gráficas más interesantes de analizar.



En general encontramos dos tipos de gráficos. El primero es como el de V36 vs V1, le cual tiene una forma triangular inversa, que es similar a los gráficos de distribución poblacional por género. Estos no parecen dar mucha información, pues la mayor variación en fechas más nuevas seguramente se genera al tener un mayor número de datos. Por otro lado, los gráficos de V15 y V17 son los ejemplos más fuertes que vemos de una relación, aunque no fuerte, entre

valores altos y fechas recientes. Esto nos da una forma casi triangular al respecto. Aunque es importante resalar un rango de fechas sin datos, que rompen las tendencias en todos los casos. Ahora utilizaremos Componentes Principales para identificar matemáticamente las variables de mayor influencia.



Encontramos que algunas de las variables previamente identificadas visualmente se comprueban matemáticamente como las de interés y de alta influencia sobre nuestra variable de interés. Además, existen otras, y que es un gran número de datos y se dificulta su fácil visualización. Por último, notamos que al usar covarianzas se dificulta la interpretación de los datos y su relación con la variable.

1.2. Selección y Extracción de variables

Antes de hacer el *feature selection*, se encontró que todas las observaciones de la variable V4 son 0, por lo tanto, desde un principio no se tuvo en cuenta este predictor.

Para la selección y extracción de variables, el proceso se dividió en 2 etapas:

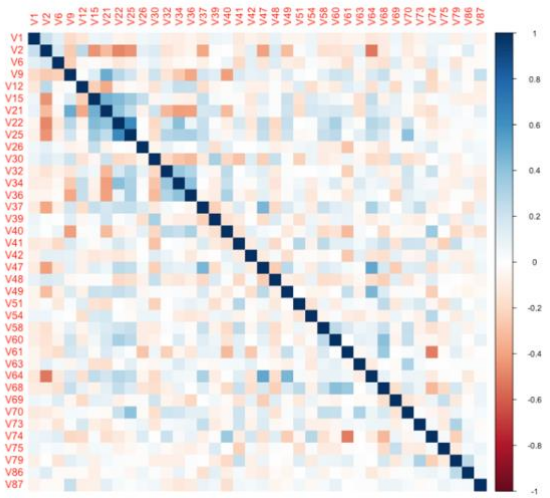
- Selección de Variables Correlacionadas a Y
- Eliminación de Variables Correlacionadas con Otras

Para la primera etapa se escogieron solo aquellas variables que cumplieran con la siguiente condición:

$$\left|\rho_{X_j,Y}\right| > 0.03$$

Una vez se evaluó la condición para cada uno de los predictores, se escogieron 37 de las variables iniciales. Con estas variables se avanzó a la segunda etapa.

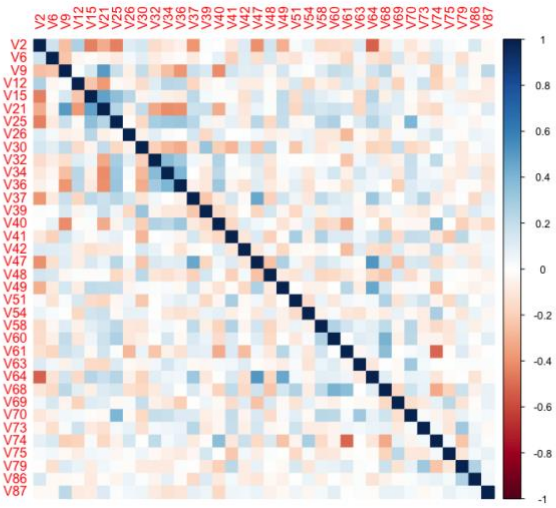
Para la segunda etapa, primero se revisó la matriz de correlaciones entre todas las variables y se encontró que hay varias con valores absolutos altos, como se puede ver en la siguiente gráfica:



Por lo tanto, para descartar estas variables se utilizó el siguiente algoritmo del libro Applied Predictive Modeling (2013) de Max Kunh y Kjell Johnson:

1. Calcular la matriz de correlaciones de los predictores
2. Determinar los dos predictores con la correlación en valor absoluta (llámelos A y B)
3. Calcule la correlación promedio de A con todos los demás predictores. Haga lo mismo para B
4. Si A tiene un promedio mayor, elimine A. De lo contrario, elimine a B.
5. Repetir pasos **2-4** hasta que no haya correlaciones mayores a un threshold determinado

Este algoritmo se utilizó usando un threshold de 0.65, y como resultado el número final de variables fue de 36 predictores. Este conjunto de variables está constituido tanto por variables de timbre como de covarianzas. La matriz de correlaciones final es la siguiente:



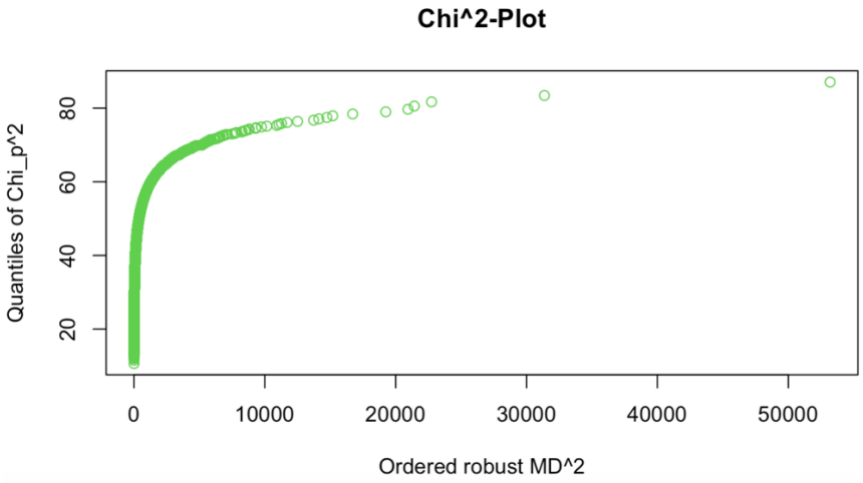
Las variables finales, de las cuales varias coinciden con los hallazgos hechos en la sección de exploración de datos, que se tuvieron en cuenta fueron:

- V2
- V6
- V19
- V12
- V15
- V21
- V25
- V26
- V30
- V32
- V34
- V36
- V37
- V39
- V40
- V41
- V42
- V47
- V48
- V49
- V51
- V54
- V58
- V60
- V61
- V63
- V64
- V68
- V69
- V70
- V73
- V74
- V75
- V79
- V86
- V87

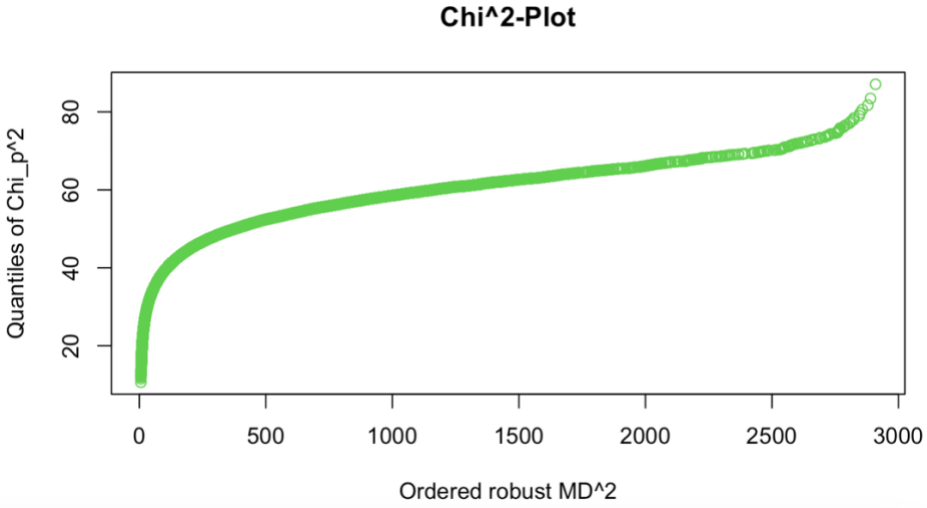
Algo importante que vale la pena recalcular sobre el proceso de selección de variables es lo curioso de la naturaleza de las variables en este problema. En términos generales, el objetivo de la selección de variables es escoger aquellas variables que más aporten información nueva al modelo, y por lo tanto se evitan las variables con información redundante. Y en este problema específico sucede algo muy interesante, porque solo 12 variables son información aparentemente independientes, mientras que las 78 variables restantes son covarianzas. Es decir 78 variables de los datos son por definición dependientes del valor de otras 2 variables. Esto hace que el *feature selection* sea especialmente complicado.

1.3. Preparación y limpieza de datos

Al revisar los datos no había *missing data*, por lo que se procedió a evaluar outliers. Usando la distancia de Malahanobis y la función *chisq.plot()* en R, sobre los datos con la selección de variables anterior, se encontraron los siguientes resultados:



En esta grafica es evidente que hay ciertas observaciones que están muy lejanas de las demás. Por lo tanto, se eliminaron las 170 observaciones con mayor distancia, y la gráfica de outliers resultante es la siguiente:



En la gráfica anterior es evidente que todavía hay bastantes observaciones que se podrían considerar outliers, pero que no fueron eliminados de los datos de train. La razón de esto tiene 2 componentes principales. El primero es que, de acuerdo con el enunciado, los datos ya venían bastante limpios, por lo tanto, no se pondrían a considerar errores de medición, sino observaciones bastante distintas a las demás. Por lo tanto, es importante capturar esta información distinta que aportan. El segundo componente viene de la naturaleza del problema. La música en general es una industria que se presta a que haya bastante varianza en las características de cada canción. A pesar de que año a año hay tendencias en los estilos musicales, siempre habrá artistas y canciones que se salgan por completo de lo que se considera *mainstream* cada año. Ejemplos de esto pueden ser artistas que hacen música retro, o inspirada en regiones de otro lado del mundo, o incluso, que están empezando un nuevo movimiento musical. Por lo tanto, estos datos *outliers* pueden ser útiles para predecir también el año de estas canciones distintas.

1.4. Evaluación de modelos

Para el problema de regresión se utilizaron 5 modelos:

- Random Forest
- Boosting
- GAM
- Red Neuronal
- SVM

Para la calibración de los primeros 4 modelos se dividieron los datos de entrenamiento en 2 conjuntos distintos. El primero era el de entrenamiento con 76000 datos escogidos aleatoriamente, y el de validación (y comparación durante calibración) con los datos restantes.

Random Forest:

Puesto que este algoritmo es resistente al número de predictores, para la calibración se usaron los datos completos, excluyendo los *outliers* identificados anteriormente. Para la calibración se probaron los siguientes hiperparametros:

- $m = \{3,6,9,15\}$
- $depth = \{2,3,4,5,6\}$
- $B = 500$

Para comparar cada uno de los modelos posibles se utilizó un $MSE_{Validacion}$ a partir de los datos de validación mencionados anteriormente. Los resultados de cada uno son los siguientes:

		depth				
		2	3	4	5	6
m	3	117.0458	115.9191	115.139	114.4055	114.0306
	6	116.2895	114.8612	113.5561	112.8428	112.174
	9	115.5747	114.1893	112.53	111.8217	111.116
	12	115.2426	113.3286	112.0199	111.0871	109.9125
	15	114.7019	112.9619	110.9518	110.145	109.3422

El modelo que mejor resultados dio entre todas las opciones fue aquel donde: $m = 15$, $depth = 6$, el cual tenía un $MSE_{Validacion} = 109.3422$. Al evaluar este modelo en Kaggle se obtuvo un $RMSE = 10.37566$

Boosting:

Puesto que este algoritmo es resistente al número de predictores, para la calibración se usaron los datos completos, excluyendo los *outliers* identificados anteriormente. Para la calibración, se probaron los siguientes hiperparametros:

- $depth = \{2,6,9,12,15\}$
- $\lambda = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$
- $B = 100$

Para comparar cada uno de los modelos posibles se utilizó un $MSE_{Validacion}$ a partir de los datos de validación mencionados anteriormente. Los resultados de cada uno son los siguientes:

		labmda					
		0	0.2	0.4	0.6	0.8	1
depth	3	119.096	84.3898	81.14636	83.62092	79.96664	87.70654
	6	119.096	79.76397	76.42013	75.44441	76.62133	87.451
	9	119.096	76.10916	72.65085	76.34595	75.11718	82.75042
	12	119.096	73.39302	66.95558	68.16846	70.94687	80.60399
	15	119.096	71.52841	66.81528	69.25336	70.82334	83.10041

El modelo que mejor resultados dio entre todas las opciones fue aquel donde: $depth = 15$, $\lambda = 0.4$, el cual tenía un $MSE_{Validacion} = 66.81528$. Al evaluar este modelo en Kaggle se obtuvo un $RMSE = 9.3127$

GAM

Puesto que este algoritmo es resistente a tener varios predictores, para la calibración se usaron lo datos completos, excluyendo los outliers identificados anteriormente. El primer modelo intentado con este algoritmo fue aquel sin ningún tipo de pre-procesamiento de datos usando todas las variables (excepto V4) y con Splines en cada una. El RMSE de ese modelo fue de 9.1881. Posteriormente se aplicó la misma fórmula a la función, con la única diferencia de que fue entrenado con los datos train **sin *outliers***, y se obtuvo $RMSE = 9.18479$. Algo interesante de este último modelo, es que al revisar el *summary()*, solo 12 variables, de las 90 en total, resultaron siendo no significativas.

Red Neuronal

Para el desarrollo de este modelo se utilizó la tecnología Keras. A diferencia de los modelos anteriores, los cuales eran independientes al número de predictores, las redes neuronales son algoritmos que pueden sufrir bastante por eso. Por lo tanto, se utilizaron los datos con *feature selection* y sin *outliers*. La arquitectura general de la solución era una red con 2 capas escondidas y una tercera de salida para las predicciones, usando como algoritmo de descenso Momentum-Based Gradient Descent .

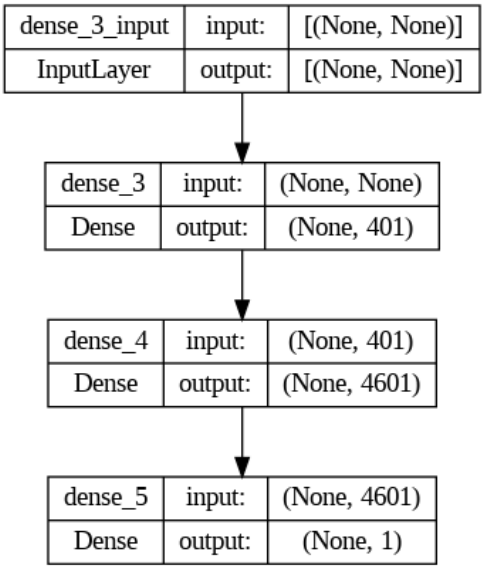
Para la calibración del modelo se siguió el tutorial *"Automatic Neural Network Hyperparameter Tuning for Tensor Flow Using Keras Tuner in Python*" (2022) del canal de YouTube Greg Hogg. Los hiperparametros probados fueron los siguientes:

- $nodos\ capa\ escondida\ 1 = \{1, 101, 201, \dots, 1000\}$
- $nodos\ capa\ escondida\ 2 = \{1, 101, 201, \dots, 1000\}$
- $funcion\ activacion = \{sigmoid, ReLu\}$

Al correr el *tuner*, se encontró que el siguiente modelo es el que tenía mejor $MSE_{Validacion}$.

- $nodos\ capa\ escondida\ 1 = 401$
- $nodos\ capa\ escondida\ 2 = 4601$
- $funcion\ activacion = sigmoid$

Es decir, usando la función *model_plot()* en Python se obtiene el siguiente modelo (el número de variables de entrada no aparece por el *tuner* sin embargo son 36):



Por lo tanto, el número total de parámetros estimables del modelo es de 1,869,041. Al evaluar este modelo en Kaggle se obtuvo un $RSME = 10.08404$

SVM

Para el desarrollo de este modelo se utilizó el paquete SVR de sklearn en Python, y de igual manera que la red neuronal anterior, se utilizaron los datos con *feature selection* y sin *outliers* para entrenarlo. Para la calibración se utilizó un set de validación con tamaño del 15% de los datos de entrenamiento, y los hiperparametros probados fueron los siguientes:

- $Kernel = Radial$
- $C = \{0.1, 0.5, 1, 2\}$
- $\gamma = \{0.01, 0.05, 0.1, 0.5, 1, 2, 5\}$

Para comparar cada uno de los modelos posibles se utilizó un $MSE_{Validacion}$ a partir de los datos de validación mencionados anteriormente. Los resultados de cada uno son los siguientes:

		C						
		0.01	0.05	0.1	0.5	1	2	5
gamma	0.1	116.231356	106.415685	103.269079	97.158626	94.5374683	91.865026	87.9136036
	0.5	128.147398	126.595158	125.10992	119.867499	118.180033	115.342935	109.709398
	1	128.509513	128.22972	127.883813	125.440538	125.15913	123.611526	119.322417
	2	128.536624	128.364905	128.151704	126.520442	127.010942	126.268715	123.155187

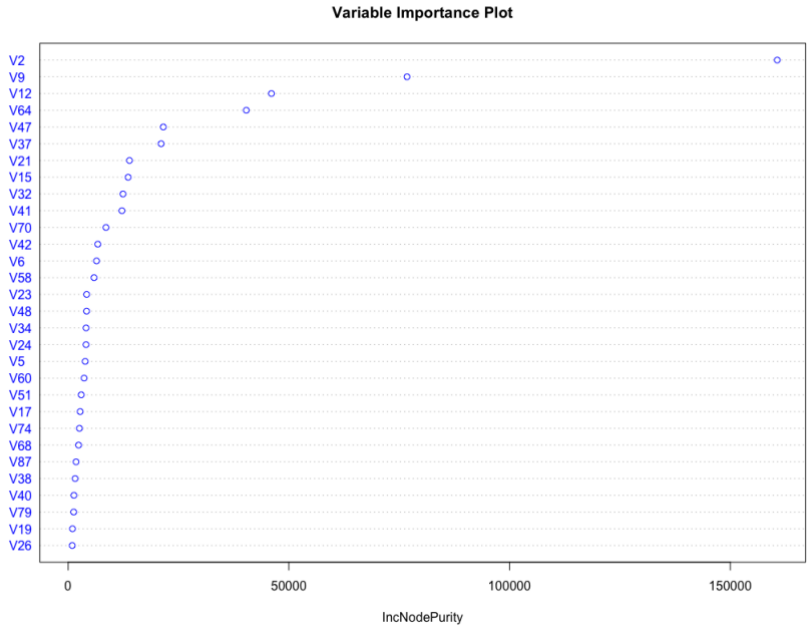
El modelo que mejor resultados dio entre todas las opciones fue aquel donde: $C = 15$, $\gamma = 0.4$. Al revisar el resultado se encontró que el modelo utilizo 65079 vectores de soporte. Posteriormente, al evaluar este modelo en Kaggle se obtuvo un $RMSE = 9.49531$.

Además, se llegó a considerar utilizar el procedimiento propuesto por Christine Dewi y Rung-Ching Cheng en 2019 en su publicación titulada *"Random Forest and Support Vector Machine On Features Selection for Regression Analysis"*. En esta publicación proponen la siguiente metodología:

1. Construir un modelo de Random Forest
2. Evaluar la importancia de cada una de las variables y escoger aquellas con mayor valor

3. Construir un SVM con dichas variables
4. Calibrar dicho SVM

Por lo tanto, utilizando el modelo de **Random Forest** calibrado anteriormente, se procedió a evaluar las significancias de cada una de las variables y se obtuvo el siguiente resultado.



En la gráfica anterior se puede encontrar las variables con mayor significancia (entre las 90 originales), y si se analiza a detalle se puede encontrar que, entre todas las visibles solo 4 no fueron escogidas como resultado del *feature selection* hecho previamente: V5, V23, V24 y V17. Teniendo en cuenta este resultado se decidió no volver a construir un modelo SVM con dichas variables ya que en la gráfica es evidente que su aporte no es particularmente alto.

1.5. Selección de modelo final, análisis y conclusiones

A partir de los valores de RMSE obtenido en Kaggle, es evidente que el mejor modelo desarrollado fue el GAM que usa todas las variables y elimina los *outliers*, aplicando Splines en cada variable. El siguiente mejor modelo evaluado fue Boosting, y en tercero puesto el VSM calibrado.

En términos generales se pueden hacer 2 observaciones interesantes sobre los resultados del problema. Primero es como cada modelo manejo la maldición de la dimensionalidad en comparación a la selección de variables hecha. Por un lado, los dos mejores modelos pertenecen a la categoría de algoritmos resistentes al número de predictores, y si nos basáramos únicamente en las significancias de retorna el modelo GAM, se podría llegar a decir que la mejor opción sería utilizar todas las variables en los modelos. Sin embargo, cuando se compararon las mayores significancias del modelo de Random Forest con el resultado del *feature selection*, la mayoría de las variables coincidían con las seleccionadas a partir del análisis basado en correlaciones. Esta coincidencia se reflejó en el resultado del SVM, el que obtuvo un RMSE bastante similar a Random Forest a pesar de llegar a las variables utilizadas de forma completamente distinta.

Segundo, fue interesante ver como se desempeñaron los modelos que utilizaban información local, información global y aprendizaje lento. No hubo una dominancia completa de ningún modelo porque GAM y Random Forest son ambos que utilizan información local y quedaron en posiciones opuestas del ranking. Y entre estos dos modelos se hay una variedad de las demás opciones, como por ejemplo aprendizaje lento y Boosting, o redes neuronales con información global.

Como conclusión del problema de regresión del proyecto, se determinó que el mejor modelo desarrollado es un GAM con todas las variables, usando Splines en cada una, y omitiendo los mayores *outliers*. Mas allá de eso se podría decir que en a través del desarrollo de este problema se pudo experimentar como los distintos algoritmos no lineales pueden ser afectados, o no, por la maldición de la dimensionalidad, y como cada uno puede llegar a depender del pre-procesamiento de los datos.

2. Problema Clasificación: Bancarrota

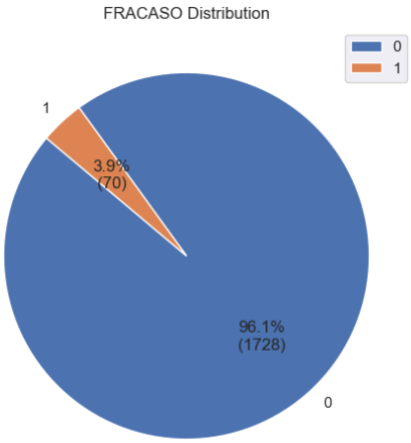
Introducción al problema:
Todos los años, empresas colombianas van a bancarrota, por lo que no son capaces de pagar sus obligaciones a corto y largo plazo. Aparte de la pérdida clara de los socios a partir de la bancarrota, también está la pérdida a considerar de las entidades bancarias a los cuales las empresas que se van a bancarrota tienen obligaciones. Es por esto, que se vuelve de suma importancia saber si una empresa se va a bancarrota o no para las decisiones de si otorgar un crédito o no.

Para manejar este problema se realizará un modelo predictivo de clasificación de aprendizaje de máquinas.

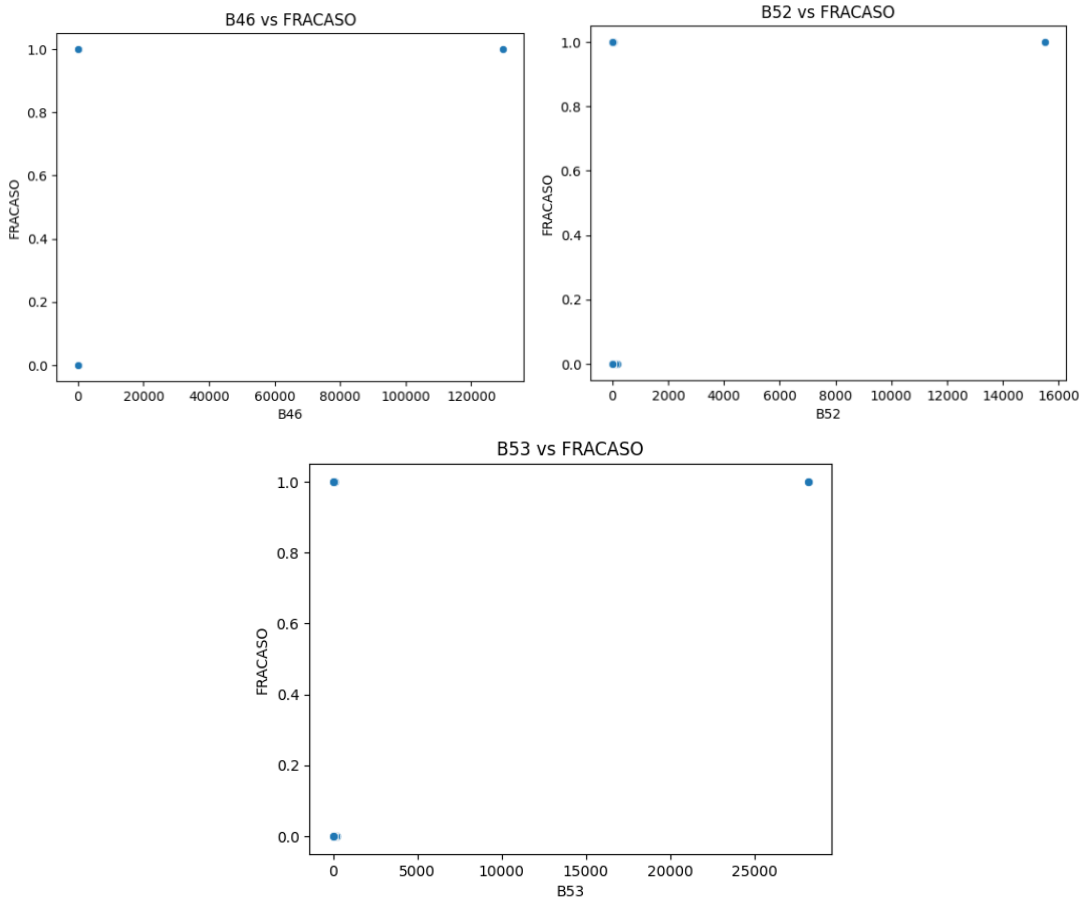
2.1. Exploración y Visualización de datos

El primer paso es entender los datos como tal. Recordemos que para la clasificación se tiene una etiqueta de FRACASO el cual es booleano y quiere decir que el valor de pasivos excede los activos. Además, contamos con 40 columnas de variables predictoras, estas son cocientes entre variables tradicionales del problema. Todos los datos son de las empresas del sector de *retail* en Colombia. Primero entendamos la estructura de los datos y la distribución de nuestra variable de interés.

Forma datos entrenamiento	(1798, 41)
Tipos	FRACASO: int64 El resto: float64
Nulos	No hay

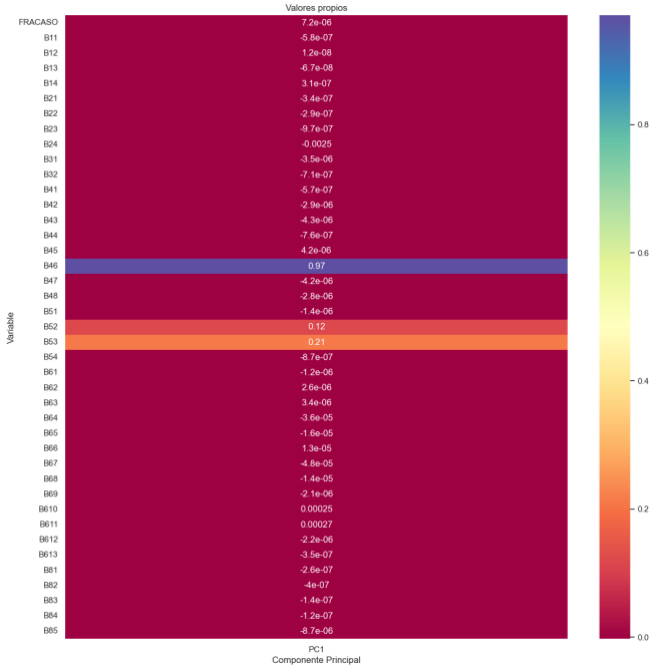


Tenemos 1798 datos de entrenamiento, donde las variables predictoras todas son float64 y no hay datos nulos. Es decir que es un *dataset* bastante bueno y limpio. Ahora bien, menos del 4% son casos de Fracaso, lo cual representa únicamente 70 datos demostrando un fuerte desbalanceo de clases, lo cual es de esperarse con la naturaleza del set de datos. Ahora procedemos a analizar las relaciones entre la variable objetivo, 'FRACASO', y cada variable por medio de un gráfico. En este caso por motivos de espacio solo incluiremos las gráficas más interesantes de analizar.

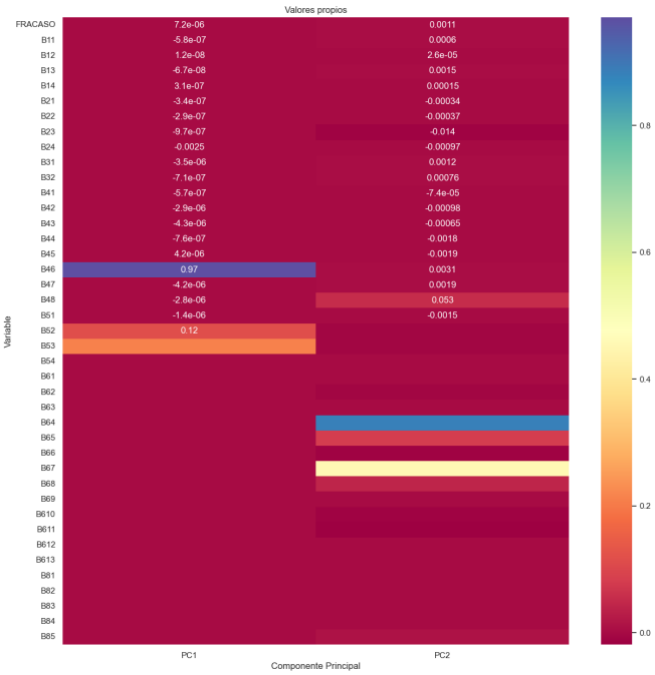


Podemos observar las gráficas de las variables B46, B52 y B53 contra la variable de interés. Estas son interesantes al tener un comportamiento casi categórico, que además predice el fracaso. Estas tres variables, nos permiten fácilmente identificar que sus valores altos siempre

son fracaso en los datos que tenemos. Es importante recalcar que nuestro interés es en la detección de fracaso, entonces, aunque hay variables con comportamiento similares pero inversos, donde un valor alto dice no-fracaso, estas no son de nuestro interés principal. Ahora utilizaremos Componentes Principales para identificar matemáticamente las variables de mayor influencia.

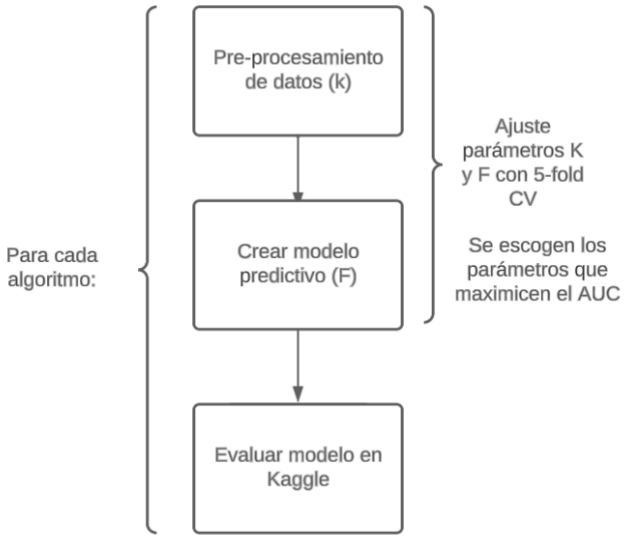


Encontramos que las mismas tres variables previamente identificadas visualmente se comprueban matemáticamente como las de interés y de alta influencia sobre nuestra variable de interés. Además, notamos que las demás variables están fuertemente no-relevantes. También podemos modificar el número de componentes de parámetro de la función de PCA para generar más componentes, donde vemos un cambio de enfoque a otras variables, pero de menor relevancia que los identificados inicialmente en el primer componente principal.



2.2 Exploración y Visualización de datos

La metodología que se usará para encontrar este modelo predictivo será de dividirlo en 3 pasos grandes:



Donde:

$$K: \{Conjunto\ de\ parámetros\ para\ realizar\ el\ preprocesamiento\ de\ datos\}$$

$$F: \{Conjunto\ de\ parámetros\ para\ crear\ los\ modelos\}$$

Consecuentemente, se realizará, para cada modelo un preprocesamiento de los datos y la creación y ajuste de cada modelo predictivo.

Para ajustar los conjuntos de parámetros K y F se realizará un gridsearch con los posibles valores óptimos de K y F, evaluando los modelos con la estrategia de 5-fold cross-validation y escogiendo y evaluando la métrica de AUC. Se realiza la predicción en TEST y, por último, se sube a Kaggle obteniendo el resultado del desempeño del modelo. A continuación, se describirá a detalle cada paso dentro de la metodología descrita anteriormente:

2.3 Selección y Extracción de Variables

Para la selección y extracción de variables, se usaron 3 grandes métodos los cuales se evaluaron con cada preprocesamiento de datos y modelos predictivos:

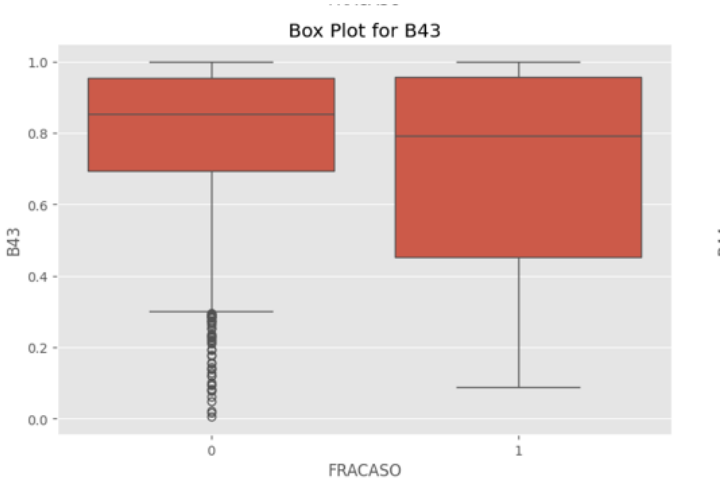
1. Selección de todas las variables
2. Selección de variables con base a boxplots
3. Selección de variables con base a la importancia dada por algoritmos basados en árboles

Como se dice anteriormente, solo hay 3 conjuntos de variables posibles, no es que se haya ajustado la cantidad de variables a usar en los dos últimos métodos descritos. Los métodos a más profundidad son:

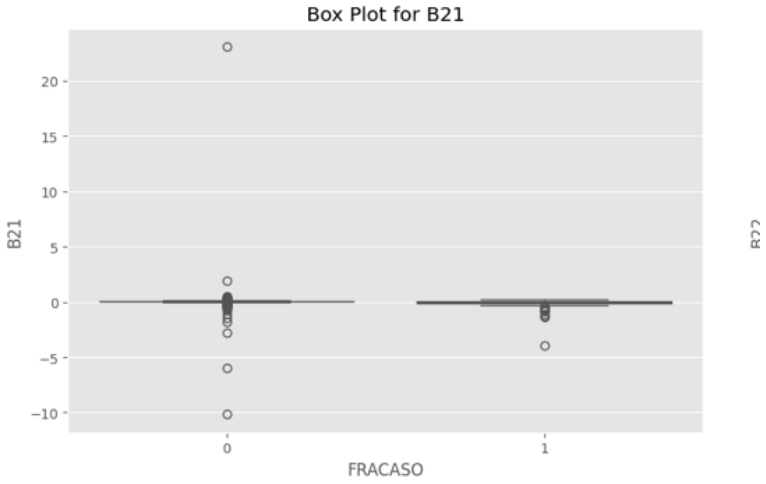
Selección de todas las variables:
En este método no hay mucho que explicar, simplemente se usaron todas las variables y no se hizo ningún tipo de selección o filtro de variables.

Selección de variables con base a boxplots:
Para este método simplemente se realizó boxplots de todas las variables predictivas con respecto a FRACASO. Al hacer esto, se dividen en 2 categorías las variables: 1. Variables con boxplots visibles y 2. Variables con boxplots no visibles. Para este método se escogieron las variables con boxplots visibles. A continuación, se muestra ejemplos de los dos tipos de categorías:

Variables con boxplots visibles:



Variables sin boxplots visibles:



En este caso, Se escogería la variable B43 y no se escogería la variable B21.

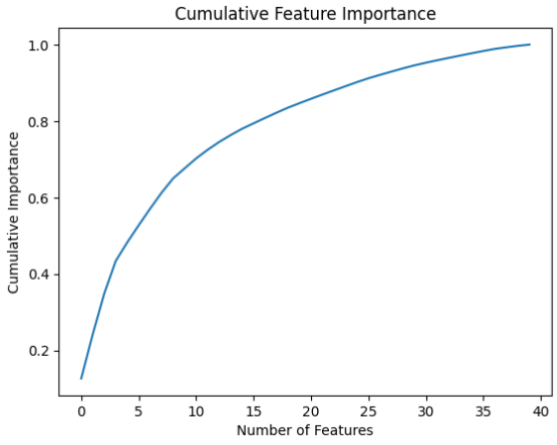
Las variables filtradas por este método fueron:

B11, B13, B14, B21, B23, B24, B46, B48, B51, B52, B53, B54, B61, B62, B63, B64, B65, B66, B67, B611, B85.

Método por importancia dada por métodos basados en ensamblaje de árboles:
Para este método se realizó el modelo Random Forest con todas las variables y después se hallaron las importancias de las variables dada por este algoritmo. El resultado fue el siguiente:

```
B21: 0.12664721068429943
B22: 0.11541074338760621
B24: 0.10653964208566367
B85: 0.0853692596622215
B45: 0.048573476440739276
B47: 0.04487751884952473
B23: 0.04370911736583264
B46: 0.04178948480042943
B67: 0.03712337712626225
B613: 0.026240900867956364
B32: 0.026008370261326345
B66: 0.022651627236049927
B31: 0.020334267143456677
B51: 0.01800513746795652
B84: 0.01669097210168923
B82: 0.014159283358440203
B48: 0.014079901895491388
B83: 0.013802766993184092
B68: 0.013160735166741416
B610: 0.011689800316462303
B54: 0.011406190708318427
B611: 0.011103412106802258
B53: 0.010960394521661428
B81: 0.010840027189864911
B44: 0.010748590467833396
B612: 0.010049620992559863
B65: 0.00884716070312264
B63: 0.008557275907210624
B52: 0.008444791497142019
B43: 0.007899354297188148
B41: 0.006885703795762443
B62: 0.006535032905591964
B64: 0.006164134486554322
B13: 0.006110798374444037
B61: 0.0059878447758219214
B14: 0.005851592562147018
B42: 0.005506000559789588
B69: 0.004106094789149206
B11: 0.0038648512269369856
B12: 0.003303534920765225
```

Esta importancia se mide entre 0 y 1. Para entender más o menos cuántas variables a escoger, se realizó una gráfica de la importancia acumulada:



Como se puede ver, se debe escoger una cantidad de variables relativamente alta, debido a que esta curva no aumenta tan rápido, lo que indica que cada variable tiene importancia relativamente alta. Para este caso, se hizo el filtro de las variables las cuales no tengan una

importancia por encima del 1%. Esto resultó en 28 variables escogidas, que fueron las siguientes:

```
['B21',
 'B22',
 'B23',
 'B24',
 'B31',
 'B32',
 'B44',
 'B45',
 'B46',
 'B47',
 'B48',
 'B51',
 'B53',
 'B54',
 'B66',
 'B67',
 'B68',
 'B610',
 'B611',
 'B612',
 'B613',
 'B81',
 'B82',
 'B83',
 'B84',
 'B85']
```

2.4 Tratamiento de Outliers

Para este paso, se utilizó la siguiente metodología: Local outlier factor algorithm con una tasa de contaminación variable y ajustable: [0.0,0.05,0.1,0.15]. Local Outlier Factor es un algoritmo de clasificación de outliers con respecto a su densidad y a la densidad de sus vecinos más cercanos (Wikipedia, s.f.).

Es decir, se realiza el LOF y se quita entre el 0% de los datos y el 15%. Este paso va dentro del pipeline de ajuste, es decir para cada modelo predictivo se encontrará la mejor tasa de contaminación que va dentro de los valores dichos.

Cabe mencionar que debido a que existe el problema de clases desbalanceadas, el quitar aún más datos por considerarlos outliers podría llegar eliminar aún más información preciada de la clase desbalanceada, por lo que este paso se tiene que hacer con mucha cautela. También se intentó eliminar los outliers detectados por LOF únicamente a ellos que sean de la clase NO FRACASO, pero esto genera sesgo dentro del modelo y no se obtuvo ninguna mejoría del rendimiento con respecto a remover los outliers independientemente de su clase.

2.5 Tratamiento de Clases Desbalanceadas

Dado que se tiene el problema de clases desbalanceadas, se corregirá por medio del algoritmo SMOTE (Synthetic minority oversampling technique). Este método se basa en crear

sintéticamente datos de la clase desbalanceada. Estos datos sintéticos estarán basados según sus vecinos más cercanos (Brownlee, 2021).

En este caso, se deja un número de vecinos estático (10 vecinos) y se ajustará el ratio entre el número de datos en clase desbalanceada con respecto a la clase no desbalanceada. Se escogieron los siguientes posibles valores de este ratio: [0.1,0.2,0.5,0.8,1].

2.6 Ajuste, Evaluación, y Selección de Modelos

Para este problema específico, se utilizaron en total 5 algoritmos de clasificación:

1. Random Forest
2. XGBoost
3. Máquinas de soporte vectorial
4. Redes neuronales
5. Regresión logística con regularización elastic net.

Se debe de tener en cuenta que para tanto para LOF como para SMOTE se evaluará el ajuste de estos 2 para cada modelo con sus hiperparámetros.

Random Forest

Random Forest es un algoritmo de aprendizaje supervisado que construye múltiples árboles de decisión durante el entrenamiento y combina sus predicciones para mejorar la precisión y reducir el sobreajuste, mediante la técnica de bagging (ensamblado bootstrap). Cada árbol se entrena con un subconjunto aleatorio de datos y predictores, y la predicción final se obtiene promediando las predicciones individuales o tomando la moda en el caso de problemas de clasificación.

Para este caso se ajustó el modelo de Random Forest sobre el número de iteraciones y sobre la profundidad máxima de cada árbol. También se ajustó el parámetro correspondiente a LOF y a SMOTE descrito anteriormente.

El mejor modelo resultante de Random Forest fue de 50 iteraciones, max_depth de 4, Smote sampling strategy de 0.1 y contamination LOF de 0. Este modelo terminó siendo el mejor de todos, obteniendo un AUC en Kaggle de 0.858, dándonos el quinto lugar en la competencia.

XGBoost

XGBoost es un potente algoritmo de aprendizaje supervisado de ensamblaje de árboles impulsados por gradiente, optimizando iterativamente la combinación de múltiples árboles para mejorar la precisión de las predicciones.
Este modelo tiene muchísimos más parámetros que su parecido -Random Forest-. Los parámetros a los cuales se ajustaron los datos fueron los siguientes:

1. Max Depth
2. Learning rate
3. Subsample
4. Colsample by tree
5. Colsample by level
6. Reg Alpha
7. Reg lambda
8. Gamma

9. SMOTE sampling strategy
10. LOF contamination

El mejor modelo resultante fue el siguiente:

11. Max Depth: 3
12. Learning rate: 0.001
13. Subsample: 0.6
14. Colsample by tree:0.5
15. Colsample by level:1
16. Reg Alpha:5.2
17. Reg lambda:3.7
18. Gamma:8.6
19. SMOTE sampling strategy: 1
20. LOF contamination: 0.05

Esto resultó en un AUC en Kaggle de 0.85, no mejor que el del Random Forest explicado anteriormente.

Máquinas de Soporte Vectorial

La Máquina de Soporte Vectorial (SVM) es un algoritmo de aprendizaje supervisado que busca encontrar el hiperplano óptimo para separar clases en un espacio multidimensional, maximizando la distancia entre los puntos de datos más cercanos de cada clase.

Para este modelo se ajustaron los siguientes parámetros:

1. C
2. Kernel

El mejor modelo resultante fue el que tenía un C de 1 y un kernel radial. Este modelo obtuvo un AUC en Kaggle de 0.835, con SMOTE de 0.7 y un LOF de 0.05. No siendo mejor que ni el Random Forest ni XGBoost.

Redes Neuronales

Para este caso se escogió una red neuronal profunda con 2 capas con 64 nodos ocultos en la primera y 128 en la segunda. Para el ajuste de esta red neuronal se hizo la escogencia de los parámetros de regularización:

1. Dropout (Ajustando la tasa de dropout)
2. Regularización Lasso (Ajustando el lambda asociado a esta regularización)
3. Regularización Ridge (Ajustando el lambda asociado a esta regularización)

El mejor modelo resultante fue de la regularización tipo dropout con una tasa de 0.1 en cada una de las capas intermedias, un SMOTE de 1 y un LOF de 0. Esto tuvo un AUC en Kaggle de 0.8342, siendo un poco menor que el de máquina de soporte vectorial con kernel radial.

Regresión Logística con Regularización Elastic Net

La regresión logística es un modelo de aprendizaje supervisado utilizado para problemas de clasificación binaria. En lugar de predecir directamente el valor de la variable dependiente como en la regresión lineal, la regresión logística estima la probabilidad de que un evento pertenezca a una categoría en particular.

La regresión logística con regularización elastic net es un modelo que combina tanto la regularización L1 (LASSO) como L2 (Ridge) para mejorar la generalización y prevenir el sobreajuste al ajustar coeficientes en la regresión logística.
En este caso, se ajusta el modelo según:

1. Alpha
2. Lambda

Esto resultó en un Alpha óptimo de 0.3 y un lambda mínimo de 0.0149. También se encontró un SMOTE óptimo de 1 y un LOF óptimo de 0.1. Esto tuvo un AUC de 0.830 en Kaggle. Lo que no fue mejor que ninguno de los modelos mencionados previamente.

2.7 Selección de Modelo Final y Conclusiones

En conclusión, el mejor algoritmo para clasificación de bancarrota fue Random Forest, con selección de variables por medio de boxplots, oversampling technique SMOTE con sampling strategy de 0.1, LOF contamination de 0, 50 iteraciones y un max depth de 4. Es interesante que el modelo menos elaborado, en cuestión de que no quitamos outliers, con pocas iteraciones y un SMOTE bajo, haya sido el mejor modelo de todos en términos de rendimiento.

Uno de los aspectos posibles a mejorar que no consideramos fue: correlación entre variables escogidas, utilizar estrategias SMOTE y LOF constantes para todos los modelos y no ajustarlas para cada modelo, diferentes técnicas de solución de desbalanceo de clases como upsampling y downsampling, y otras técnicas diferentes para tratar outliers. También se puede considerar la partición de la muestra en train y test, donde train se divide en train y validation para hacer la estrategia de k-Fold cross-validation para el ajuste de hiperparámetros y cuando se tenga el modelo resultante evaluarlo en TEST para no utilizar muchos intentos en Kaggle que son limitados. En total se subieron 23 intentos a Kaggle, donde lamentablemente no se pudo subir del quinto escalafón.

3. Referencias

- Hogg, G. (2022). *Automatic Neural Network Hyperparameter Tuning for Tensor Flow Using Keras Tuner in Python.* Recuperado de: https://www.youtube.com/watch?v=6Nf1x7qThR8
- Kuhn, M., Kjell, J. (2013). *Applied Predictive Modeling*. [PDF]
- Dewi, C., & Chen, R. C. (2019). Random forest and support vector machine on features selection for regression analysis. *Int. J. Innov. Comput. Inf. Control*, *15*(6), 2027-2037.
- Brownlee, J. (17 de Marzo de 2021). *SMOTE for Imbalanced Classification with Python*. Obtenido de Machine learning Mastery: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
- Wikipedia. (s.f.). *Local outlier factor*. Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Local_outlier_factor