Minería de datos: Aspectos Avanzados

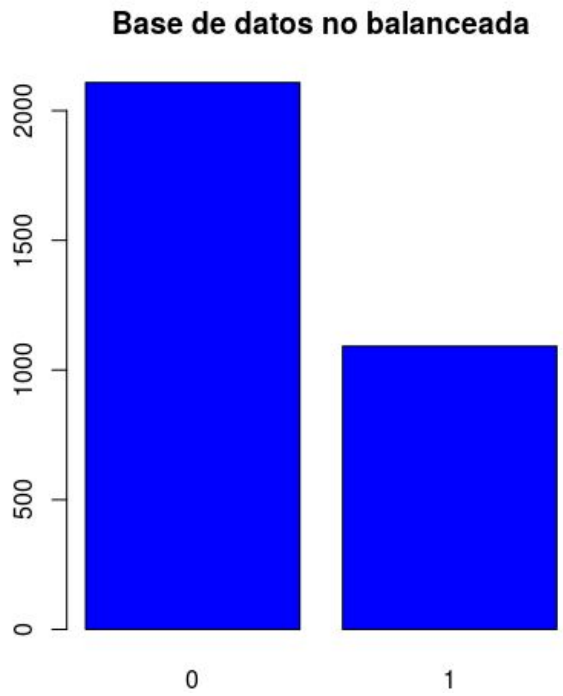
Clasificación No Balanceada

Máster en Ciencia de Datos e Ingeniería de los Computadores

Besay Montesdeoca Santana Graduado en Ingeniería Informática
Nombre de usuario de Kaggle: BesayMontesdeocaSantana

1. Descripción de la base de datos

La base de datos con la que trabajaremos en esta competición consta de 22 atributos numéricos con dos clases(0 y 1) y unas 3200 instancias. La base de datos presenta un ratio de desequilibrio de aproximadamente el 50% (IR = 0.5180266):



2. Experimentación con los modelos y con las técnicas de balanceo.

Antes de entrar de lleno en el entrenamiento de los modelos y en la aplicación de las técnicas de balanceo de los datos, es necesario presentar algunas metodologías y estrategias que se han implementado para luego realizar el aprendizaje de los modelos.

Se hizo una selección de características de las variables de nuestra base de datos en primer lugar, para evitar tener que estar tratando con variables que no aportan información para la tarea de aprendizaje, ya que en teoría algunos algoritmos de aprendizaje funcionarán mejor (más rápido, con menos memoria) y además serán más fácilmente interpretables al ser más simples. Para realizar esta tarea nos apoyamos en el paquete llamado FSelector el cual dispone de una serie de algoritmos capaces de realizar una primera extracción de características e identificar las variables, en un principio, más relevantes de cara a la clasificación en algunos modelos:

```
# Función que simplifica el uso del paquete FSelector para la extracción de características
featureSelector = function(dataBase, n){
 weights <- FSelector::chi.squared(Class~.,dataBase)
 sub = FSelector::cutoff.k(weights, n)
 f = as.simple.formula(sub, "Class")
 print(f)
 return(f)
}
```

En segundo lugar es necesario explicar cómo se realizan las particiones que se utilizarán luego en el entrenamiento de los modelos. Dichos modelos deberían ser evaluados sobre un conjunto de muestras no usadas ni para aprender el modelo ni para ajustar sus parámetros. Esta sería la única forma de obtener una medida de eficiencia sin sesgo. En esta práctica se utilizó la técnica conocida como *k-validación cruzada*. Esta forma de validación implica el particionado del conjunto de datos en k subconjuntos de igual tamaño (aproximadamente) y obtener k modelos diferentes dejando una partición, de forma sucesiva, como conjunto de test. Es decir, en todos los modelos se usan k − 1 particiones para entrenamiento y 1 para test. La estimación final ofrecida sería la media de las estimaciones para cada uno de los k modelos:

```
# Parámetros
k.cv = 10
dataBase = balancedData
classPosition = 23
classFormula = formula(Class ~ .)
classFormula = f

# Se crean las particiones del conjunto de datos
ind = seq(1,nrow(dataBase),by=1)
trainPartitions = createFolds(ind, k = k.cv, returnTrain = TRUE)

# Generamos de la misma forma las particiones de test
testPartitions = list()
for(i in 1:k.cv){
 testPartitions[[i]] = ind[-trainPartitions[[i]]]
}
```

Por último quedaría presentar cómo se evaluará el rendimiento de los modelos. En esta práctica se utilizara el área bajo la curva ROC(AUC), el cual se podría interpretar como la probabilidad de que un clasificador ordenará o puntuará una instancia positiva elegida aleatoriamente más alta que una negativa. Para ello utilizaremos el paquete de R llamado pRoc donde utilizaremos una de sus funciones (auc) para evaluar cada iteración que se ha entrenado con una partición para luego finalmente realizar la media de todas las k particiones.

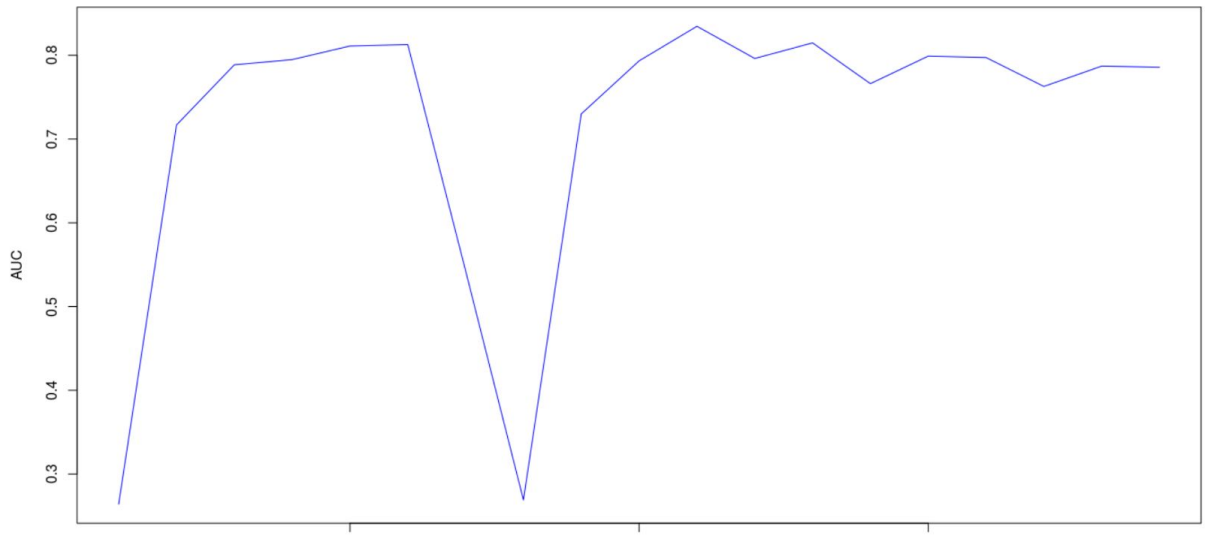
En un principio no se hizo hincapié en las técnicas de balanceo para mejorar el rendimiento de la clasificación, sino que se procedió a comprender y familiarizarse con los distintos modelos de clasificación. Por tanto se ejecutaron varias veces los algoritmos de árbol de decisión, Random Forest, svm, boosting y bagging donde se fueron variando y analizando los distintos parámetros para comprender su funcionamiento.

Una vez que se hubo familiarizado con el uso y ajuste de los parámetros de los distintos modelos, se procedió a probar las distintas técnicas de balanceo así como a averiguar qué combinaciones con los modelos ofrecen mejores resultados.

En esta práctica se ha utilizado el paquete *unbalanced* de R para probar las distintas técnicas, donde se ha experimentado con tres: SMOTE, Tomek Link y ENN.

Los resultados y las pruebas más representativas se reflejan en la siguiente tabla:

Modelo	Tec. Balanceo	Parámetros	AUC. Local	AUC Kaggle
Arbol Dec	SMOTE	-	0.887284	0.26419
randomForest	SMOTE	ntree = 5	0.981065	0.71673
randomForest	SMOTE	ntree = 100	0.995166	0.78856
randomForest	SMOTE	ntree = 1000	0.995793	0.79476
randomForest	Tomek link	ntree = 1000	0.871354	0.81101
randomForest	ENN	ntree = 1000	0.854823	0.81266
svm	SMOTE	cost = 10, gamma = 5	0.998556	0.54264
svm	SMOTE	cost = 3, gamma = 0.1	0.992907	0.26909
svm	SMOTE	cost = 3, gamma = 0.1, Extracción de características	0.903562	0.72996
svm	Tomek link	cost = 3, gama = 0.1, Extracción de características	0.861658	0.79325
svm	Tomek link	cost = 3, gama = 0.001, Extracción de características	0.84984	0.83452
svm	Tomek link + SMOTE	cost = 3 gamma = 0.001, Extracción de características	0.83346	0.79611
svm	ENN	cost = 3 gamma = 0.001, Extracción de características	0.829384	0.81473
boosting	SMOTE	mfinal = 20, maxdepth = 5, minsplit = 5	0.937353	0.76607
boosting	tomek link	mfinal = 20, maxdepth = 5, minsplit = 5	0.864215	0.79890
bosting	ENN	mfinal = 20, maxdepth = 5, minsplit = 5	0.845538	0.79725
bagging	SMOTE	mfinal = 20, maxdepth = 5, minsplit = 5	0.839015	0.76272
bagging	Tomek Link	mfinal = 20, maxdepth = 5, minsplit = 5	0.820063	0.78702
bagging	ENN	mfinal = 20, maxdepth = 5, minsplit = 5	0.803909	0.78566



En cuanto a las técnicas de balanceo se podría afirmar que la que mejor rendimiento dió para este problema fue la técnica Tomek Link, ya que parece ser que la estrategia de limpiar la frontera entre clases es la más adecuada. Sin embargo la técnica SMOTE, donde como sabemos para cada ejemplo de la clase minoritaria introduce ejemplos sintéticos en la línea que une al elemento con sus 5 vecinos más cercanos, no ofrece tan buenos resultados como se esperaba posiblemente porque pudo haber creado malos ejemplos que posteriormente confundieron los clasificadores. Por último la técnica ENN, donde se eliminan las conjuntos ruidosos del conjunto original en función de sus vecinos, se puede observar que ofreció rendimientos similares a los obtenidos con la técnica SMOTE.

Con respecto a los clasificadores, en general se podría decir que todos los modelos ofrecen rendimientos similares, pero si debemos destacar uno, el que mejor predicciones ha realizado es el modelo basado en Máquinas de Soporte Vectorial (SVM).

Para probar los distintos modelos con las diferentes técnicas de balanceo y con el objetivo de no implementar codigo redundante, se implementó un bucle donde se recorren las distintas k particiones y dónde se iba comentando y descomentando aquellas partes del cogido para probar los modelos:

```
measure = c()
# Bucle de generacion de modelos
for(i in 1:k.cv){
 cat("k =", i, "\n")

 # Modelos

 # model = ctree(classFormula, dataBase[trainPartitions[[i]], ])
 # testPred = predict(model, newdata = dataBase[testPartitions[[i]], ], type = "prob")
 # testPred = sapply(1:length(testPred), function(n){testPred[[n]][1]})

 model = svm(classFormula, data = dataBase[trainPartitions[[i]], ],cost=3, gamma = 0.001, probability
= T)
 testPred = predict(model, newdata = dataBase[testPartitions[[i]], ], probability = T)
 testPred = attr(testPred,"probabilities")[,1]
```

```
 # model = randomForest(classFormula, dataBase[trainPartitions[[i]], ], ntree = 150)
 # testPred = predict(model, newdata = dataBase[testPartitions[[i]], ], probability = T)
 # testPred = predict(model, newdata = dataBase[testPartitions[[i]], ], type = "prob")[,2]
 #
 # model = adabag::boosting(classFormula, data = dataBase[trainPartitions[[i]], ],
 #                          mfinal = 20,
 #                          control = rpart::rpart.control(maxdepth = 5, minsplit = 5))
 # testPred = predict(model, newdata = dataBase[testPartitions[[i]], ])$prob[,1]

 # model = adabag::bagging(classFormula, data = dataBase[trainPartitions[[i]], ],
 #                         mfinal = 20,
 #                         control=rpart::rpart.control(maxdepth= 5, minsplit= 5))
 # testPred = predict(model, newdata = dataBase[testPartitions[[i]], ])$prob[,1]

 # Calculamos el área bajo la curva ROC (AUC)
 aucM = auc(dataBase[testPartitions[[i]], classPosition], testPred)[1]
 cat("AUC =", aucM, "\n\n")
 measure = cbind(measure, aucM)
}

finalMeasures = apply(measure, 1, function(n){sum(n)/k.cv})
cat("\n\nAverage AUC Measures\n")
print(finalMeasures)
```

3. Conclusion

En resumen, se han investigado los distintos modelos de clasificación y se han analizados distintas las técnicas de balanceo que mejor se ajustan al problema. Después de este pequeño estudio se podría afirmar que la mejor configuración de algoritmos fue la técnica de balanceo Tomek Link junto con el modelo SVM. El mejor modelo se entrenó con coste igual a 3 y un gamma de 0.001. En cuanto al kernel se ha optado por el polinómico donde el tipo de la predicción fue C-classification. Por último destacar que el modelo se entrenó con una selección de las 10 mejores variables de la base de datos y con un escalado automático de las mismas.