Reconocimiento de Patrones

Trabajo Final

Reconocimiento de Letras

8 de diciembre 2014

Leticia Grassi Johnny Silvera

Índice de contenido

Introducción

En este trabajo se presenta un posible enfoque para el diseño de clasificadores automáticos para el caso de clases desbalanceadas. Con este fin se nos fue propuesto un problema de reconocimiento de letras en el cual el archivo de trainning presenta 26 clases desbalanceadas. En el transcurso de este trabajo se plantearan métodos de selección de características, balanceo de datos y clasificación estudiando su performance contra un archivo de test.

Para la segunda parte se procederá a estudiar un caso en el cual se cuenta con la información que el archivo de test proviene de un texto en español y se buscara la forma de hacer pesar esta información sobre los métodos seleccionados.

Objetivo

Dado un conjunto de datos de testing donde N_j es la cantidad de muestras de la clase j y E_j la cantidad de elementos de esa clase mal calificados por un algoritmo dado, la tasa de acierto para la clase j, A_j viene dada por $A_j=\left(1-\frac{E_j}{N_J}\right)$.

La media aritmética (Amean) de las tasas de acierto por clase se define como $Amean=\frac{1}{C}\sum_{j=1}^{C} A_j$ donde C es el numero de clases para este caso C=26.

Se desea lograr un sistema que sea capaz de clasificar correctamente y de forma automática a que letra corresponde cada vector de características del conjunto de test. El requerimiento mínimo a alcanzar es que la media aritmética de las tasas de acierto por clase ese por encima de 90%.

Parte 1- se asume equiprobabildad entre las clases

Parte 2 – se establece como nuevo objetivo que el sistema maximice su tasa de acierto, conociendo que el conjunto de testeo proviene de un texto en idioma español.

Datos

Se tiene una base de datos basados en imágenes en blanco y negro de 26 letras en mayúscula del alfabeto. A partir de estas imágenes se tiene 16 características, sobres las que se trabajara.

Se dispone de un conjunto de muestra de entrenamiento previamente etiquetadas en la forma (vector característica, clase). Este consta de 16000 elementos. Y un conjunto de test que consistirá en 4000

elementos extraídos de un texto en español (tomando la ñ como n).

Se gráfica la cantidad de patrones por clase

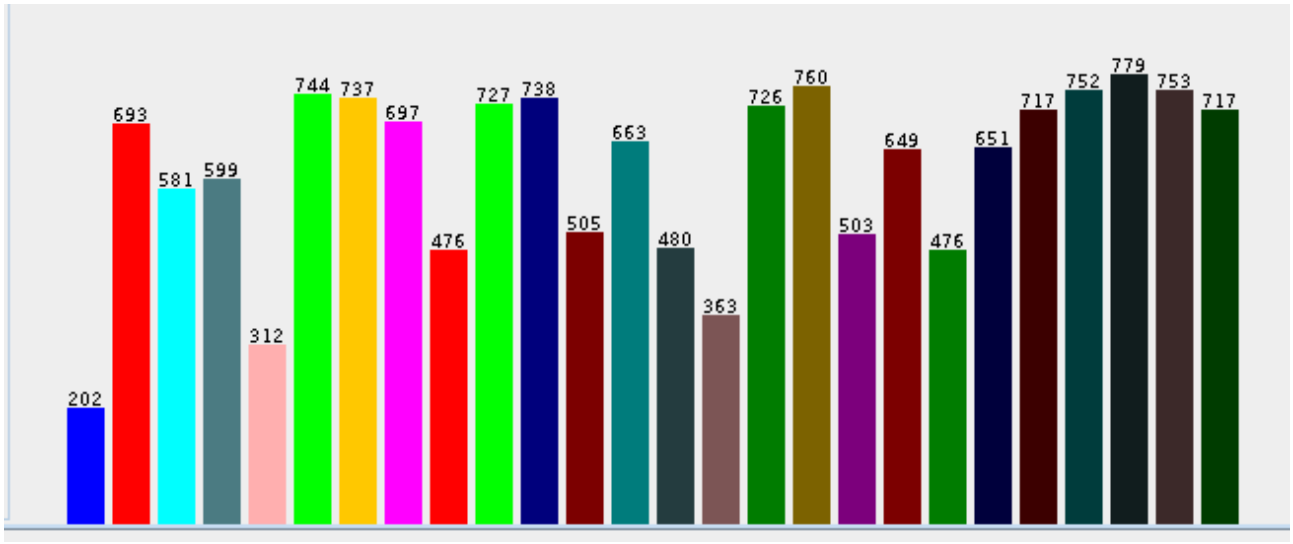

Figura1- Cantidad de patrones por clase

De la figura 1 podemos ver que las clases no están balanceadas y letras como la “X” y “Q” son lo tienen mayor cantidad de patrones mientras la “A” y “E” las de menor cantidad.

Sección y Extracción de Característica

Se realizo un estudio previo de selección de características utilizando distintos métodos, sin balanceo de las clases. Estas técnicas nos permiten extraer información relevante de los datos, por medio de eliminación de redundancia, ruido, o disminuir la dimensión del problema.

Método de Ganancia de Información

Evalúa el peso de un atributo midiendo la ganancia de información con respeto a una clase

$$InfoGain(Class, Attribute) = Entropia(Class) - Entropia(Class \mid Attribute)$$

Ranking de atributos obtenidos:

	Ranked	Attributes
1	0,9239	13
2	0,8161	11
3	0,7957	14
4	0,7795	7
5	0,7655	12
6	0,7465	15
7	0,7171	9
8	0,6515	8
9	0,599	10
10	0,4775	6
11	0,4126	16
12	0,1248	3
13	0,115	5
14	0,0774	1
15	0,0464	4
16	0	2

Tabla 1: Atributos seleccionados infoGain

Método de razón de Ganancia de Información

Evalúa el peso de un atributo midiendo la razón de ganancia de información respecto a una clase

$$GainR(Class, Attribute) = \frac{(Entropia(Class) - Entropia(Class|Attribute))}{Entropia(Attribute)}$$

Ranking de atributos obtenidos:

	Ranked	Attributes:
1	0,315	13
2	0,3091	14
3	0,2587	11
4	0,2565	12
5	0,2528	7
6	0,2509	15
7	0,2212	9
8	0,1973	8
9	0,1888	10
10	0,1823	4
11	0,1638	6
12	0,158	16
13	0,0809	3
14	0,0622	5
15	0,0481	1
16	0	2

Tabla 2: Atributos seleccionados GainRatio

Componente Principales - PCA

PCA crea nuevas característica tomando combinación lineal de las características originales, trasformando el espacio en base a los mejores componentes principales los cuales representan la dirección de mayor varianza de las características originales

Se obtuvo las siguientes característica:

	Ranked	attributes:
1	0,7319	1
2	0,5628	2
3	0,4506	3
4	0,3719	4
5	0,3052	5
6	0,2468	6
7	0,1899	7
8	0,1509	8
9	0,115	9
10	0,0847	10
11	0,058	11
12	0,0415	12

Tabla 3: Extracción de atributos PCA

Se gráfico la varianza que aporta cada componente

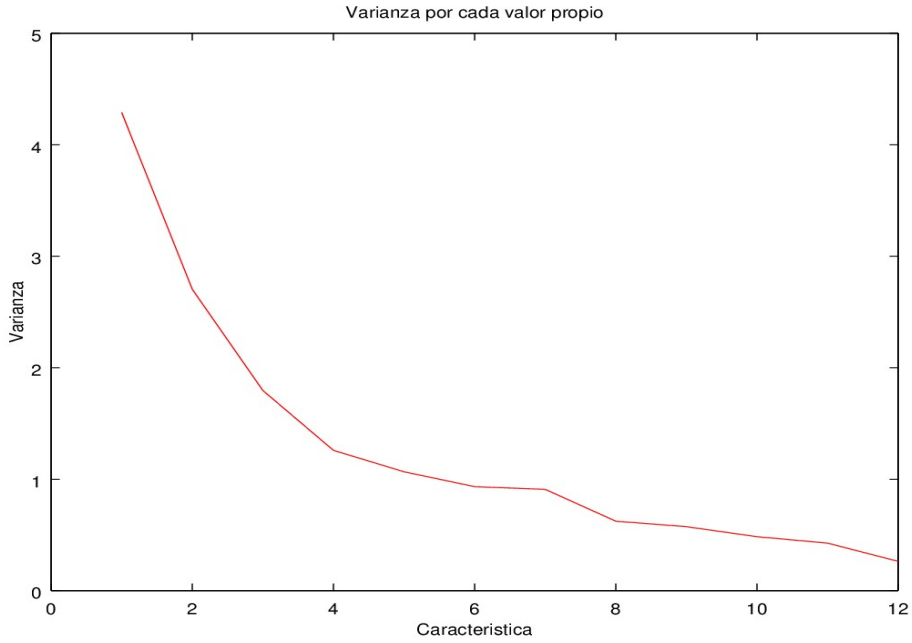

Ilustración 1: Varianza por Componente PCA

Según la gráfica anterior se puede ver como el aporte de cada nueva característica es significativa, sin poder descartar a simple vista alguna.

También se gráfico la varianza acumulada por cada características

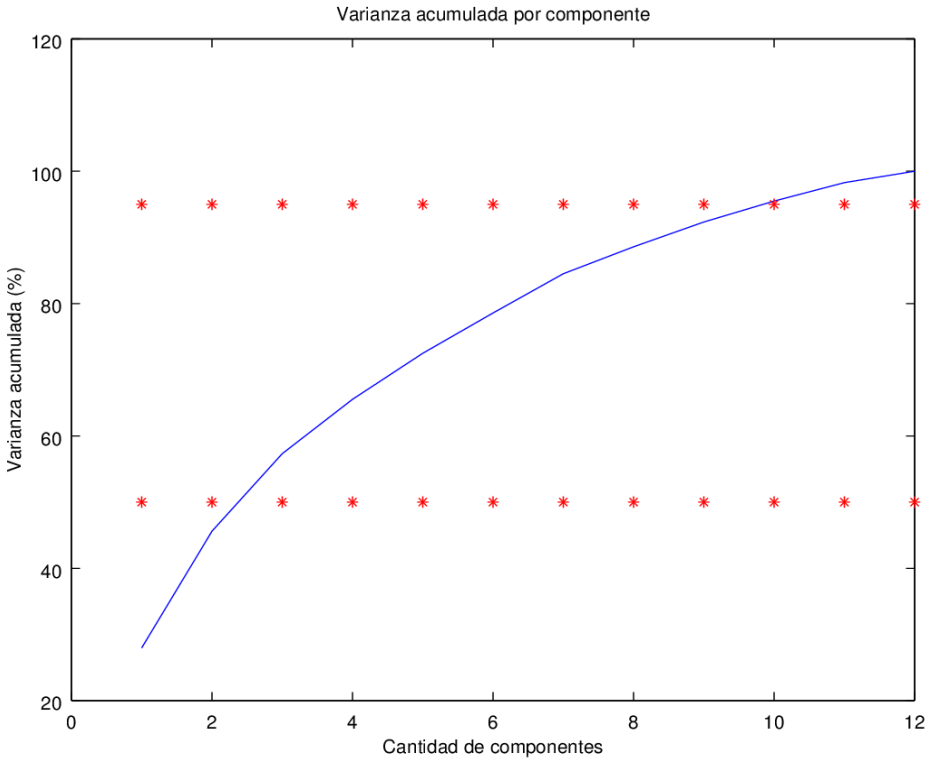

Ilustración 2: Varianza acumulada por componente

De la anterior se concluye que para tener un 95% de la varianza acumulada nos debemos quedar con por lo menos 10 característica, de forma de no perder información relevante para clasificar.

Clasificadores

Teniendo en cuenta que la media aritmética sobrepase el 90%, se utilizara distintos clasificadores, evaluando el desempeño de las características seleccionadas. Los algoritmos que mejor se acerquen al umbral pedido, se realizara un optimización de los parámetros y estudio de robustez.
Para los siguientes casos se utilizo cross-validation o split (con 70% entrenamiento y 30% de test) de las muestras de entrenamiento.

Clasificador : Naive Bayes

Método de aprendizaje supervisado basado en el teorema de Bayes, este supone independencia entre pares de muestras. Este clasificador se caracteriza por ser sumamente rápido.

Para este método se realiza una tabla comparativa donde se varia el paramento “Kernel” en WEKA para la estimación de la densidad de probabilidad, se obtuvieron los siguientes valores de media aritmética en %

NaiveBayes	Gain-Ratio	Info-Gain	PCA
Sin Kernel	64,044	64,625	65,43
Con kernel	74,275	74,275	70,11

Tabla 4: Media aritmética según selección de Características-NaiveBayes

NaiveBayes exhibe un pobre desempeño respecto a los umbrales pedidos, pero se logra mejorar el resultado al aplicar Kernel con InfoGain, Ratio y PCA. Notando una peor performance al aplicar PCA.

Clasificador: K-vecinos

Este método de clasificación supervisada que estima la función de densidad de probabilidad o directamente a la probabilidad posterior de que un elemento x que pertenezca a la clase W_j a partir de un conjunto de entrenamiento.

Llamados “lazy learning” ya que la función se aproxima solo localmente y todo los cálculos son diferido a la clasificación.

La fase de entrenamiento del algoritmo consiste en almacenar los vectores característicos y las etiquetas de las clases de los ejemplos de entrenamiento. En la fase de clasificación, la evaluación de un elemento de test, es representado por un vector en el espacio característico. Se calcula la distancia entre los vectores almacenados y el nuevo vector, y se seleccionan los k vecino más cercanos. El nuevo dato es clasificado con la clase que más se repite en los vectores seleccionados.

La mejor elección de k depende fundamentalmente de los datos; generalmente, valores grandes de k reducen el efecto de ruido en la clasificación, pero crean límites entre clases parecidas. Un buen k puede ser seleccionado mediante una optimización de uso.

Para este método se realiza un tabla comparativa entre los distintos algoritmos de selección de característica, aplicando para K en 2 ,5 y 10, obteniendo la siguiente media aritmética en %:

KNN				
number	**Sin selector Caract**	**Info gain**	**Gain Ratio**	**PCA**
2	95,45	95,093	94,12	93,012
5	94,9	94,91	94,906	93,8
10	93,85	93,881	93,9	92,77

Tabla 5: Selector de características para KNN

Se puede observar como mejora los aciertos por clase en comparación con lo obtenido con NaiveBayes, en todos los casos ; Para k en 5 o 10 vecinos la performance se mantiene sobre nivel de umbral pedido, por lo que se obtiene un buen clasificador de los datos. Al aplicar selectores de características no se notan grandes mejorías de la performace del método.

Clasificador: J48

Weka implementa C4.5 a través del clasificador J48, mediante el confienceFactor (factor de confianza utilizado en la poda) y el minNumObj (cantidad mínima de patrones por nodo para crear un hoja) realiza el podado del árbol.

Este método genera arboles de decisión para clasificación a partir de datos de entrenamiento, es fácil de interpretar, de rápida clasificación y se le puede incluir conocimiento a priori.

Se corrió J48 en weka con distintos métodos de podado, obteniendo los siguientes valores de media aritmética en %

Confidenc Factor	Sin selector De Caract		GainInfo		GainRatio		PCA	
	C4.5	Error-Prunning	C4.5	Error-Prunning	C4.5	Error-Prunning	C4.5	Error-Prunning
0.5	86.23658	82.60414	87.2125	83.4563	87.2375	83,475	78.1625	73,925
0.25	86.18633	82.60414	87.1188	83.4563	87.15	83,475	78.1625	73,925
0.1	86.05422	82.60414	86.9125	83.4563	86.9438	83,475	78.1125	73,925
0.01	85.82678	82.60414	86,675	83.4563	86,675	83,475	78	73,925

Tabla 6: Media aritmética de J48 según selector de características variando ConfidencFactor

Aun variando el Confidence Factor los valores obtenido no sobrepasan el valor de umbral, para los distintos selectores de características

Clasificador: J48 Graf [1]

Este método agrega nodos en un árbol ya existente con el objetivo de reducir el error de predicción. El algoritmo identifica regiones ocupadas por patrones mal clasificados y considera ramificaciones alternativas para las hojas de la región en cuestión.

Los parámetros variados fueron el Confidence Factor, obtenido los siguientes resultados de la media aritmética en %

Confidence Factor	csfSubseteval	GainInfo
	C4.5	C4.5
0.5	87.8	87.45
0.25	87.6563	87.3438
0.1	87.5	87.1125
0.01	86.8875	86.8

Tabla 7: Selector de característica con J48 Graf

Los valores obtenidos no sobrepasan el umbral pedido para los distintos selectores de características.

Clasificador: Random Forest [2]

Este algoritmo genera un bosque de arboles, donde se combinan arboles predictores y cada árbol depende de un subconjunto aleatorio de patrones de entrenamiento. Para clasificar un dato, cada árbol generado brinda su clasificación de forma independiente, y cada clasificación es un voto. El algoritmo selecciona la clasificación que mas votos obtenga.

Se corrió en Weka este algoritmo con parámetros por defecto, para distintos selectores de características se obtuvo la siguiente media aritmética en %

	Media aritmetica
Cfssubseteval	93.5938
Gain ratio	93,875
Infogain	93.8125
PCA	88.9625

Tabla 8: Selector de características con Random Forest

Para ese método se obtiene valores por encima del umbral pedido para Gain y ratio Info. Obteniendo una menor performance que K-NN.

Meta Clasificador: AdaBoost M1 [3]

Antes de comenzar con adaboot veamos a que nos referimos con Boosting, éste busca combinar a través de varias iteraciones de un mismo algoritmo de base las predicciones hechas de manera adaptativa. Más precisamente la performance del predictor de la etapa m, influirá sobre la manera en que se considera la muestra en la etapa m+1.

Realizando variaciones adaptativas de la muestra y asignando un peso a cada predictor intermediario se obtiene un clasificador final más eficiente, que disminuye la varianza y también el sesgo.

El algoritmo Adaboost desarrollado por Freund et Schapire[3] es sin duda el algoritmo del tipo Boosting más conocido y estudiado. Este utiliza por lo general los arboles de decisión como modelos de base. En la iteración m del algoritmo, siguiendo cierta distribución, se construye un árbol y se observa las predicciones de éste sobre todos los datos de la muestra. A aquellos datos mal clasificados, se les asignan un peso mayor, influyendo de esta manera sobre la distribución de los datos de la etapa m+1 y forzando al predictor correspondiente a tomarlas “más en cuenta”. El predictor que se obtiene al final resulta de un voto mayoritario ponderado o de un promedio

ponderado de los predictores de las distintas etapas. Al focalizarse sobre los ejemplos mal clasificados, el error empírico disminuye rápidamente. Sin embargo, en vez de contribuir a un sobre aprendizaje sobre los datos utilizados, se prueba que el error de generalización disminuye también, lo cual hace de Adaboost un algoritmo con un muy buen rendimiento.

A continuación se describe explícitamente Adaboot para la clasificación binara:

1) si $L=\{(X_1,Y_1),(X_2,Y_2),..,(X_n,Y_n)\}\ con\ X_i \in X\ YI \in \{-1,1\}$ se inicializan los pasos de las observaciones con $w_1(i)=\frac{1}{n}\ \forall i=1,..n$

2) Para m=1,..M

a. Se construye a partir de L y de los pesos $w_m(i)$ un clasificador $gm: X \rightarrow \{-1,1\}$ que minimiza el error global $\varepsilon_m=\sum_{i=1}^{n} w_m(i) 1_{g_m(xi)\neq y_i}$

b- Calculamos $\alpha_m=\log(\frac{1-\varepsilon_m}{\varepsilon_m})$

c- Actualizamos los pesos $w_{m+1}(i)=w_m(i)\exp(\alpha_m * 1_{g_m(xi)\neq y_i})$ para i=1,..n y los normalizamos

3)- El clasificador final es:

$$f(x)=\underset{y}{Argmax}\sum_{m=1}^{M}\alpha_m\{1_{g_m(xi)\neq y_i}\}=signo(\sum_{m=1}^{M}\alpha_m g_m(x))$$

Observar que a cada etapa m del algoritmo, se le asigna un peso mayor a los datos mal clasificados por g_m . Si el error ε_m , la suma de los pesos de las observaciones mal clasificados por el clasificador de la etapa m es mayor que 1/2 se detiene el algoritmo. Esta condición implica que el clasificador debe predecir mejor que una clasificación aleatoria.

Para el problema de varias clases, se conocen varias extensiones de Adaboost. Por ejemplo :

Adaboost.M1 que es el mismo algoritmo que Adaboost pero considerando que la variable dependiente pueda pertenecer a mas de dos clases.

Se corrió en weka el algoritmo adaboost con J48 y RandomForest de forma de mejorar los datos obtenido anteriormente obteniendo la siguiente media aritmética en %

J48	
Confidence Factor	**Media aritmetica**
0.5	94.7375
0.25	94.7938
0.1	94.6688
0.01	94,725

Tabla 9: Mejora de J48 con Adaboost

Random Forest	Media aritmetica
Normal	95.6
Resampling	96,106

Tabla 10: Mejora de Random Forest con Adaboost

Se observa una mejora del métodos J48 al pasar de 86% a 94% en su media aritmética, también se obtiene una notoria mejora con Random Forest pasa de 93% a 96% prácticamente. Obtenido así un buen método en el cual mejora la performance de algoritmos bases (en este caso arboles)

Clasificador : SVM

Modelo de aprendizaje supervisado no paramétrico, que construye fronteras permitiendo particionar el espacio de características de modo que se pueda asociar cada región a una respectiva clase.

Dado un conjunto de entrenamiento podemos etiquetar las clases y entrenar clasificador SVM para construir un modelo que prediga la clase a la que pertenece la nueva muestra. Mas formalmente, SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta que se utiliza tanto en problemas de clasificaron como de regresión. Logrando una buena separación de las clases permitirá una buena clasificación de los datos de test.

Veremos para este método una forma de estimar los parámetros “óptimos”.

Comenzaremos revisando el método SMO de weka: el cual es un método multiclase que resuelve los problemas de clasificación de a parejas 1 vs 1. Por lo que cada clase es comparada contra otra hallando un clasificador SVM. Son métodos de costo computacional muy alto.

Al aplicar el clasificador con los parámetros por defecto obtenemos:

SVM	
Parametos	Por defecto
Media aritmetica	81.6938

Tabla 11: SVM parámetros por defecto

Dado los resultados se hace evidente la necesidad de buscar los parámetros óptimos para este método.

Para esto basándonos en [4] se decide utilizar una búsqueda sobre una grilla con un kernel RBF.

Se arma la grilla de forma que el eje “x” sea la variable C (pesos sobre los errores) del algoritmo mientras que el “y” haga referencia a Gamma del kernel.

Realizaremos una evaluación del parámetro C de SMO de 1 a 16 con un paso de 1 y el parámetro Gamma del kernel RBF en los valores 10^-5, 10^-4,...,10^2.

Si los resultados obtenidos no se encuentran sobre el borde de la grilla serán considerados como buenos, en caso contrario se moverá la grilla de forma de poder seguir iterando hasta obtener un a solución satisfactoria.

Resultados:

weka.classifiers.meta.GridSearch

==========
Options: -E ACC -y-property classifier.kernel.gamma -y-min -5.0 -y-max 2.0 -y-step 1.0 -y-base 10.0 -y-expression pow(BASE,I) -filter weka.filters.AllFilter -x-property classifier.c -x-min 1.0 -x-

max 16.0 -x-step 1.0 -x-base 10.0 -x-expression I -sample-size 100.0 -traversal COLUMN-WISE -log-file "/home/johnny/Escritorio/Patrones/Trabajo final/paramSVM" -S 1 -W weka.classifiers.functions.SMO -- -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.RBFKernel -C 250007 -G 0.01"

Result of Step 2/Iteration 1:
[6.0, 1.0]

Como se puede observar los resultados no se encuentran en el borde de la grilla, a continuación evaluaremos con 10 folds cross-validation los parámetros obtenidos:

SVM	
Parametos	Gamma=1, C=6
Media aritmetica	94.3375

Tabla 12: SVM parámetros GridSearch

Se observa que los resultados superan de forma satisfactoria el umbral de 90% establecido por el problema por lo que se pudo optimizar los parámetros de SVM obteniendo mejor valores de clasificación.

Estudio de robustez

Se realizo un estudio de robustez de los métodos de mejor performance; Adaboost con Random Forest (resampling), KNN (InfoGain) y SMV, donde se utilizo un 70% de las muestras para entrenamiento y un 30% para test, obteniendo así la siguiente media aritmética en %:

	Adaboost – Random Forest	KNN	SVM
Seed	**Resampling**	**5 vecinos**	**C=6 G=1**
1	94.8333	93.7083	93.4583
2	95.0625	93,875	93.7083
3	95.1042	93,375	93.7917
4	95.0833	93.9375	93.7917
5	95.2292	93.8333	93.75
6	96.0813	93.8958	93.6667
7	96	94	94.0625
8	95.9688	93.2083	93.9375
9	96.2313	94	94.0625
10	96.1	93.6042	94.1458
mu	**95.5694**	**93.7437**	**93.8375**
sigma	**0.2993**	**0.0734**	**0.0453**
peor	**94.8333**	**93.2083**	**93.4583**

Tabla 13: Estudio de robustez clases des balanceadas

Para los tres métodos la varianza obtenida es muy pequeñas lo que nos garantiza una estabilidad en los métodos utilizados.

Para observar al estabilidad de los métodos de forma visual se gráfica para cada métodos su media con intervalos de confianza al 95%.

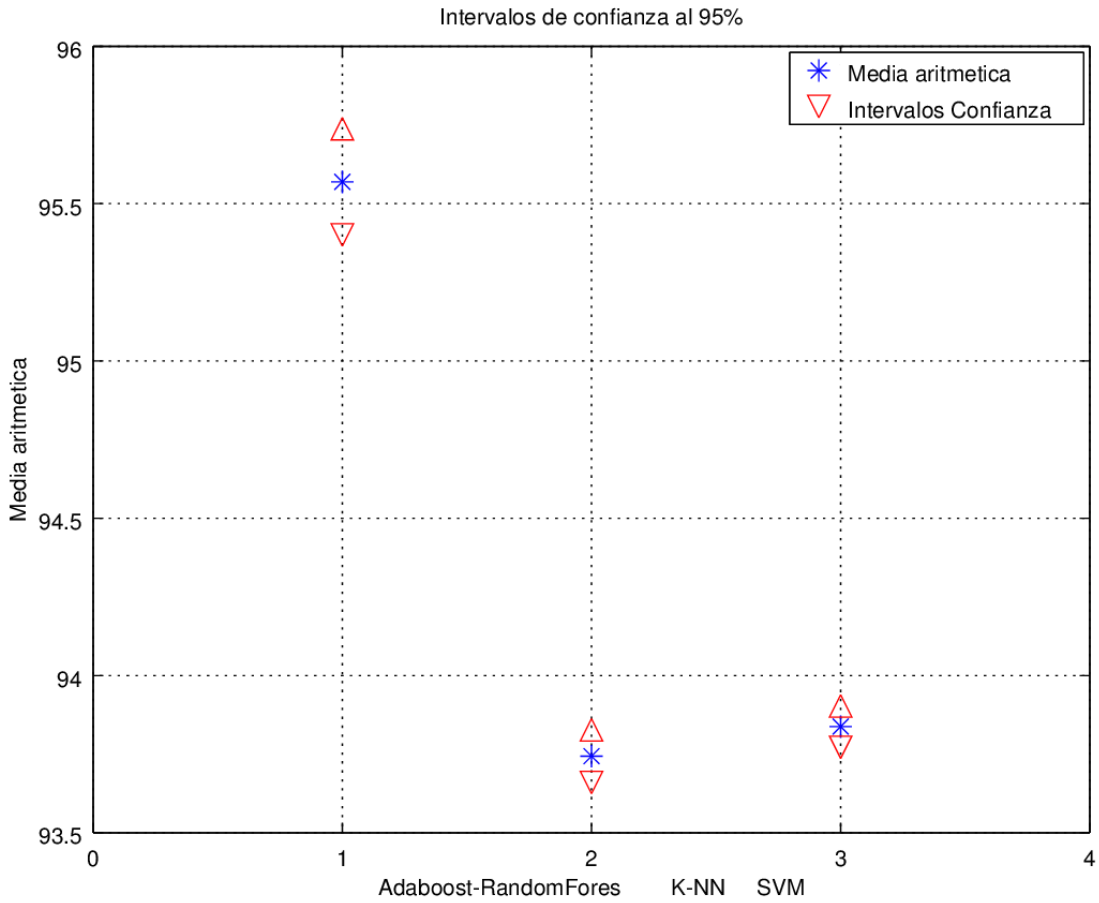

Ilustración 3: Intervalos de confianza al 95% para métodos de mejor performance

Comparando entre los tres métodos Random Forest es el que tendría mas dispersión en los valores obtenidos, pero al tener varianza tan pequeñas podemos garantizar la estabilidad de los métodos estudiados.

Conclusión

Se estudio distintos selectores de características y se evaluó su performance con distintos métodos de clasificación. A partir de estos métodos se aplicaron algoritmos de optimización (como adaboost o gridsearch) mejorando la performance obtenida.

No se obtuvo grandes mejoras utilizando selectores de características, principalmente al aplicar PCA se nota una degradación en la performance de los métodos. Algoritmos como PCA nos permite disminuir la dimensión de grandes espacio de características en base a la varianza acumulada que aporta cada una de ellas, en nuestro caso el espacio de dimensión de características es pequeño, y cada característica aporta información relevante al conjunto de entrenamiento por lo que era de esperar este algoritmo no mejorar la performance de los clasificadores utilizados. También no se obtuvieron grandes mejorías al aplicar selectores de características como infoGain y GainRatio sobre métodos de KNN o arboles.

Al aplicar el meta clasificador Adaboost en los distintos métodos base (arboles) se pudo aumentar notoriamente la performance de los métodos, esta mejoría se da ya que este método de optimización esta pensado para que en el paso m de la iteración dale mayor pesos a la datos mal clasificados de forma que en el paso m+1, el siguiente árbol predictor construido se basara en estos pesos y así disminuir el error de clasificación sin aumentar el sobre entrenamiento del árbol.

En base al estudio de robustez se encontró que algoritmos como K-NN, Adaboost con Random Forest y SVM mostrando buena estabilidad de la clasificación, obteniendo así valores de media aritmética por encima del umbral pedido.

Balanceo de Clases

Al tener clases des-balanceadas, la probabilidad a priori de cada clase esta restringida por la cantidad de muestras que tiene cada clase, haciendo que a la hora de clasificación ciertas clases tengan mas peso en comparación con otras.

Para esta parte se balanceo las muestras con dos de los métodos que proporciona WEKA, SMOTE y Resample.

Resample

Muestreo con reposición creando muestras repetidas de forma aleatoria. Este método combina las técnicas de over-sampling y under-sampling. Weka dispone de dos parámetros;

Bias- si el valor es 0, conserva la relaciona entre cantidades de clase, tal como esta el conjunto originalmente, en cambio si es 1 aplica uniformidad entre las clases obtenido clase balanceadas.

Size- tamaño final del conjunto a obtener, como porcentaje del conjunto original

Para nuestro caso se intento compensar el desbalanceo de clases con resample (con Size=100% y bias=1) donde se descartaron muestras de las clases mayoritarias y incrementado muestras de las clases minoritarias, se opto por estos parámetros con la idea de tener un under-sampling pequeño de forma de minimizar la perdida de información sobre las clase mayoritarias. Obtenido el siguiente balanceo de características.

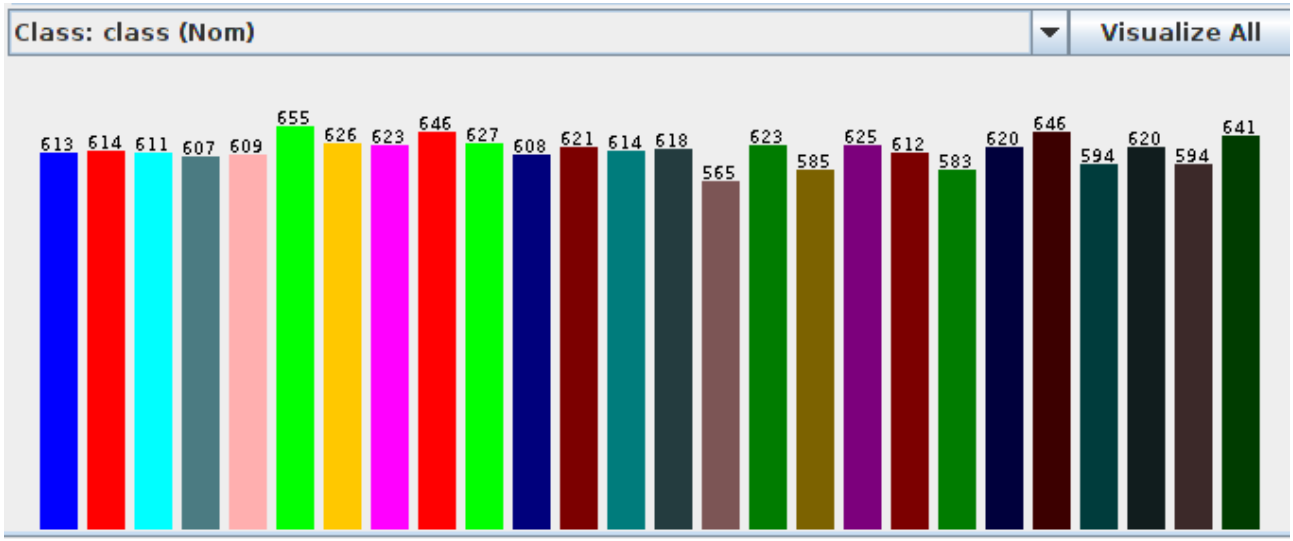

Ilustración 4: Clases balanceadas con resample

Se corrió en Weka los mismos métodos de clasificaron que en la parte des-balanceadas obteniendo los siguientes resultados

	Sin selector De Caract		InfoGain		GainRatio		PCA	
0,25	C4.5	Error-Prunning	C4.5	Error-Prunning	C4.5	Error-Prunning	C4.5	Error-Prunning
J48	91,237	86,8813	91,5063	87,112	91,437	87,069	87,425	80,669
J48 Graft	91,456	-	91,656	-	91,587	-	87,687	-
Random Forest	96,8	-	96,863	-	96,837	-	94,631	-

Tabla 14: Arboles, clases balanceadas

NAIVEBAYES				
KernelEstimator	Sin selector Caract	Info gain	Gain Ratio	PCA
False	63.6063	63.6063	63.6063	64.6625
true	74.3688	74.3688	74.3688	70.65

Tabla 15: NaiveBayes, clases balanceas

ADABOOST			
		Random forest	
Arboles	J48	Sin Resampling	Con Resampling
Sin selector De Caract	97,35	97,437	97,825

Tabla 16: Ababoost para distintos arboles

KNN				
number	Sin selector Caract	Info gain	Gain Ratio	PCA
2	95.7688	95.7938	95.8	95.0938
5	94.3125	94.2938	94.3313	93.4375
10	93.2313	93.2	93.15	92.7125

Tabla 17: KNN para clases balanceadas

SVM	
Parametos	Gamma=1, C=6
Media aritmetica	94.9813

Tabla 18: SVM para clases balanceadas

Estudio de robustez

Se realizo un estudio de robustez de los métodos de mejor performance: Adaboost con Random Forest (resampling), KNN y SMV, para esa parte se utilizo un 50% de las muestras para entrenamiento y un 50% para test para intentar disminuir el efecto del sobre entrenamiento sobre los resultados, obteniendo así la siguiente media aritmética en %:

	Adaboost Random Forest	KNN	SVM
Seed	Resampling	5 vecinos	C=6 g=1
1	95.7125	93.0417	92.65
2	95.7125	92.7292	93,025
3	95.5875	92.3958	92.85
4	95.85	93.1042	93.65
5	95.5	93.4167	93.6625
6	95.8	93.6042	93.3125
7	95.65	93.3333	92,825
8	95.7625	92.7083	93.1125
9	95.5375	92.5833	92,625
10	95,475	93.7292	92.8625
mu	**95.6588**	**93.0646**	**93.0575**
sigma	**0.0169**	**0.2055**	**0.1416**
peor	**95.4750**	**92.3958**	**92,625**

Tabla 19: Robustez de los métodos de mejor performance

Para los tres métodos la varianzas obtenida es muy pequeña lo que nos garantiza una estabilidad en los métodos utilizados.

Para observar al estabilidad de los métodos de forma visual se gráfica para cada métodos su media con intervalos de confianza al 95%.

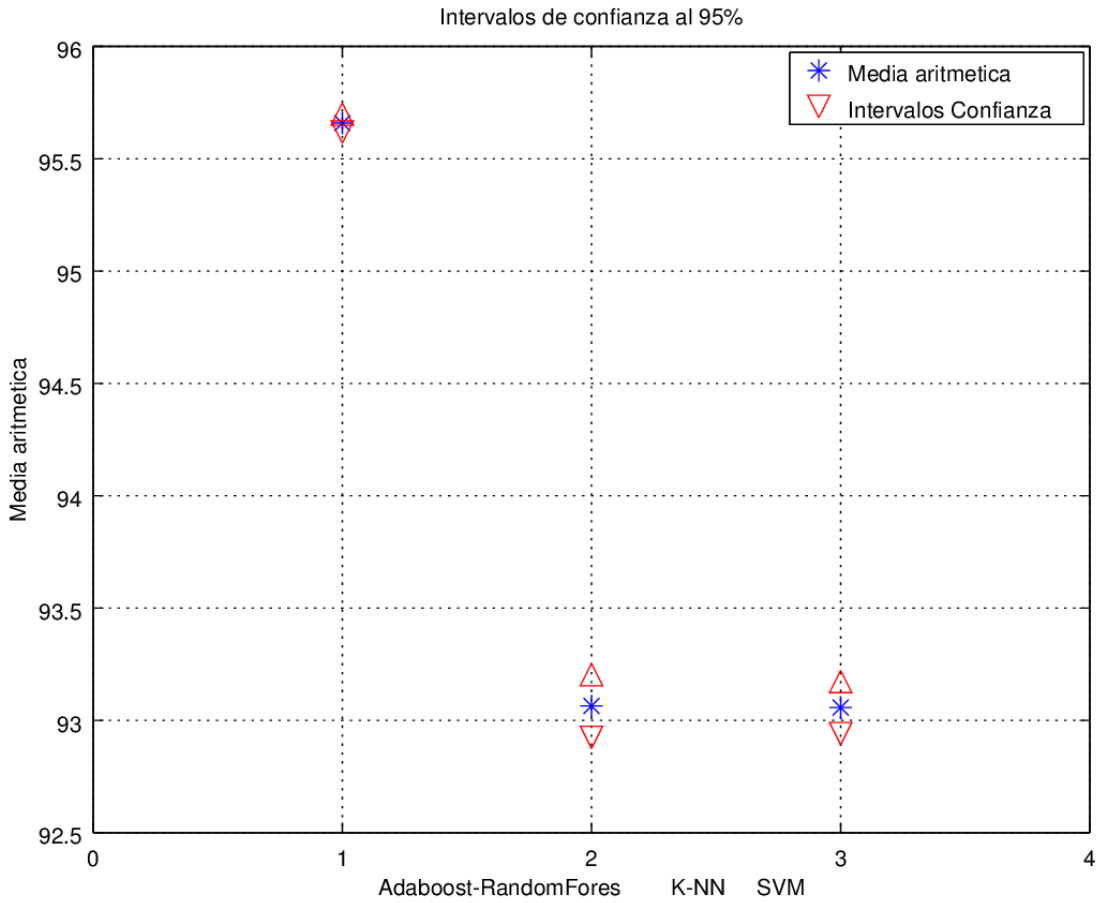

Ilustración 5: Intervalos de Confianza al 95%

Comparando las tablas 13 y 19 se observo que al balancear las clases la estabilidad y la media del método Random Forest mejora, subiendo 0,1% su media y bajando su varianza de 0,3 a 0,016, en cambio KNN disminuye su performance donde su media disminuye en un 0,7% y su varianza aumenta de 0,07 a 0,2 . En SVM también disminuye su perfomance bajando su media aritmética un 0,8%, y aumentando su varianza en 0,1; Debemos tener en cuenta que los valores de varianza son muy pequeños por lo que aun así con disminución de la performance estos métodos siguen siendo buenos clasificadores.

Conclusión

Al utilizar resample con reposición se logra balancean las muestras, introduciendo peso sobre ciertas muestras, principalmente las minoritarias y quitando información de las mayoritarias, a su vez los pesos introducidos son distribuidos de forma aleatoria pudiendo introducir ruido a los datos.

Métodos como K-NN que clasifican muestras según al cercanía a sus vecinos, pueden ser afectados por estos métodos de balanceo, ya que por ejemplo se estaría duplicando puntos en el plano de decisión haciendo que la performance de éste disminuya (generando sobreentrenamiento).

SVM clasifica en base a un hiper plano separador que maximiza el margen entre clases, este hiper

plano se halla en base a puntos de fronteras entre las clases, por lo que al utilizar resample algunos datos sera duplicados, lo que variaría el hiperplano separador, llevando a regiones de decisión mas pequeñas y especificas provocando sobre entrenamiento.

Random Forest clasificar en base a cantidad de votos que generan distintos arboles predictores independiente, cada árbol predictor es entrenado con cierto porcentaje de muestras de entrenamiento y testado con el resto, por lo que al tener muestras repetidas se podría estar entrenado y testando con las mismas muestras provocando sobre entrenamiento.

SMOTE – Syntetic Minority Over-Sampling Technique –

Este algoritmo genera instancias “sintéticas o artificiales” para equilibrar la cantidad de datos. Las nuevas muestras se generan realizando interpolación de las muestras originales, favoreciendo principalmente a las clases minoritarias.

Al aplicar este método, se debe tener en cuenta que las verificaciones como validación cruzada, o split pueden no estar actuando de forma adecuada, ya que las muestras nuevas son combinación de otras introduciendo ruido.

Weka a través de SMOTE permite generar instancias sintéticas de a una clase, por medio del parámetro “classValue” en el cual se elige la clase y con “percentage” la cantidad de instancias a generar. Se tomo como tope crear instancias hasta 150% mas de las originales con el objetivo de obtener clases balanceas y evitar grandes cantidad de muestras sintéticas.

Obteniendo la siguiente distribución de muestras:

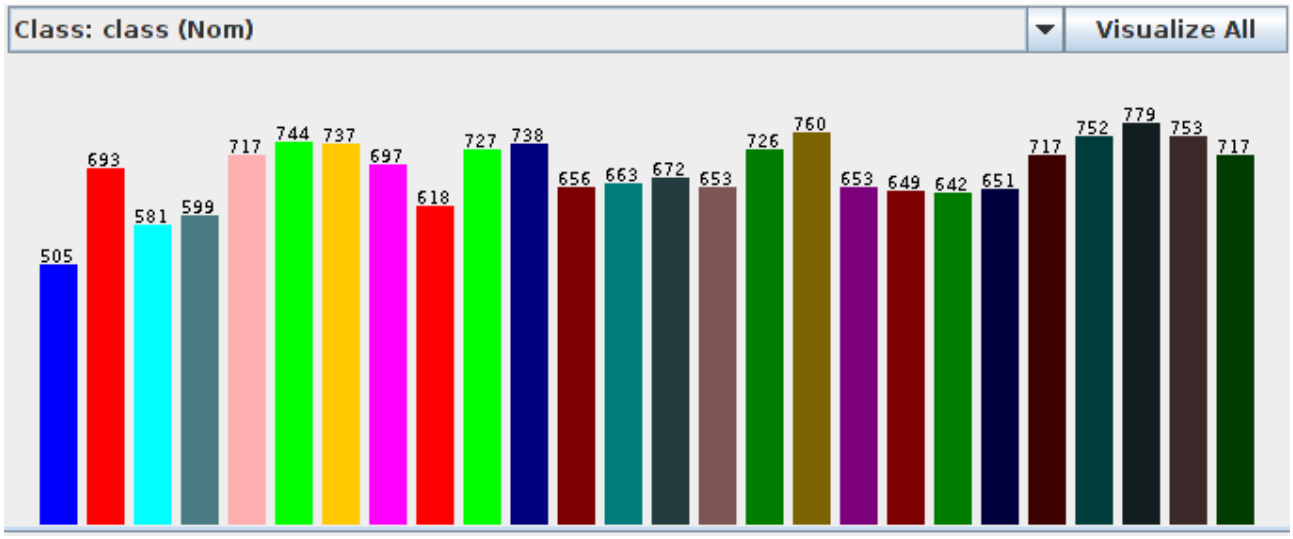

Ilustración 6: Balanceo con SMOTE

En base a las muestras obtenidas se corrió para distintos métodos de clasificación, obteniendo la

siguientes medias aritméticas en %:

NAIVEBAYES				
KernelEstimator	Sin selector Caract	Info gain	Gain Ratio	PCA
False	63.7508	63.7508	63.7508	64.9643
true	72.8917	72.8917	72.8917	70.4197

Tabla 20: Naive bayes - Balanceo SMOTE

ARBOLES								
Confidenc Factor	Sin selector De Caract		InfoGain		GainRatio		PCA	
0,25	C4.5	Error-Prunning	C4.5	Error-Prunning	C4.5	Error-Prunning	C4.5	Error-Prunning
J48	88.0499	84.1733	88.3083	84.3811	88.2802	84.3587	80.0157	75,493
J48 Graft	88.3758	-	88.6117	-	88.6061	-	80.5214	-
Random Forest	94.5615	-	94.5334	-	94.9323	-	90404	-

Tabla 21: Arboles - Balanceo SOMTE

KNN				
number	Sin selector Caract	Info gain	Gain Ratio	PCA
2				
5	95.6964	95.7245	95.7526	94.7975
10	94.6795	94.7244	94.7244	94.0165

Tabla 22: K-NN- Balanceo SOMTE

ADABOOST			
		Random forest	
Arboles	J48	Sin Resampling	Con Resampling
Sin selector De Caract	95.3593	96.3593	96.6515

Tabla 23: Adaboot- Balanceo SOMTE

SVM	
Parametos	Gamma=1, C=6
Media aritmetica	94.8256

Tabla 24: SVM- Balanceo SOMTE

Estudio de Robustez

Se realizo un estudio de robustez de los métodos de mejor performance Adaboost con Random Forest (resampling), KNN (infoGain) y SMV, donde se utilizo un 50 % de las muestras para entrenamiento y un 50% para test, obteniendo así la siguiente media aritmética en %:

	Adaboost – Random Forest	KNN	SVM
seed	**Resampling**	**5 vecinos, InfoGain**	**C=6,gamma=1**
1	94.8084	92.4935	93.3026
2	95.0556	93.2015	93.37
3	94.8534	92.5497	92.7745
4	94.9994	93.0891	93.1678
5	94.8534	93.37	93,415
6	95.0556	93.5161	93.4937
7	94.6623	92,516	93.2801
8	94.8421	93.0217	93.2464
9	94.7972	93.7072	93.4824
10	95.0893	93.2352	93.6285
mu	**94.9017**	**93.0700**	**93.3161**
sigma	**0.0197**	**0.1836**	**0.0546**
peor	**94.6623**	**92.4935**	**92.7745**

Tabla 25: Estudio Robustez- Balanceo SMOTE

Para los tres métodos la varianzas obtenida es muy pequeña lo que nos garantiza una estabilidad en los métodos utilizados.

Para observar al estabilidad de los métodos de forma visual se gráfica para cada métodos su media con intervalos de confianza al 95%.

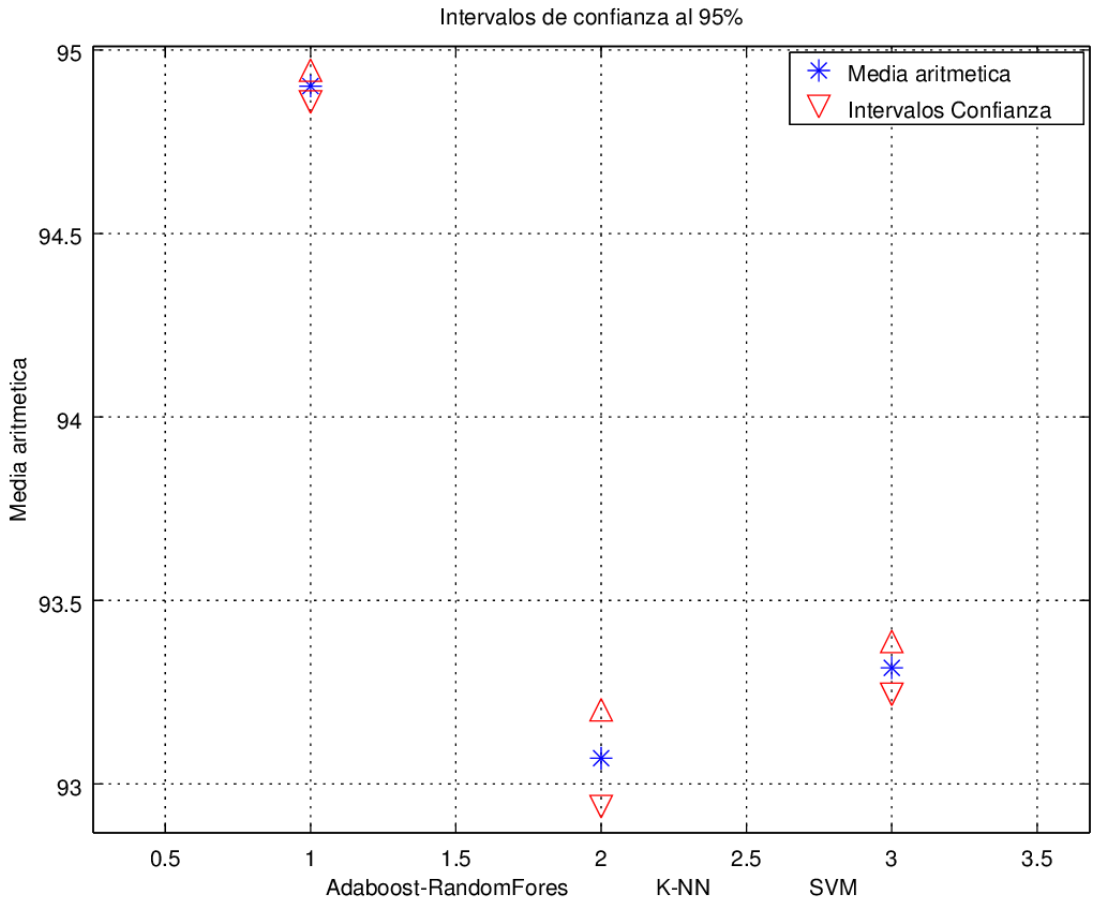

Ilustración 7: Intervalos de confianza al 95% -Balanceo SMOTE

Comparando las tablas 13 y 25 se observo que al balancear las clases con SMOTE el método Random Forest se degrada, disminuyendo 0,6% su media y bajando su varianza de 0,3 a 0,019, también para el método KNN disminuye su performance donde su media disminuye en un 0,7% y su varianza aumenta de 0,07 a 0,18 . En SVM también disminuye su perfomance bajando su media aritmética un 0,5%, y aumentando su varianza en 0,01; Estas disminuciones de performance se deben a que estos métodos son mas sensibles al ruido introducido por el SMOTE. Debemos tener en cuenta que los valores de varianza son muy pequeños por lo que aun así con disminución de la performance estos métodos siguen siendo buenos clasificadores.

Conclusiones

SMOTE introduce nuevas muestras a partir de la combinación lineal de las originales, llevándonos a concluir los siguiente para los distintos métodos utilizados

Para SVM el echo que dependa únicamente de algunos vectores de soporte lo hace relativamente robusto ante ruido y muestras que presentan desbalanceo no pronunciado. Por lo que se podria esperar una mejora en los resultados.

Para KNN el echo de que las nuevas muestras sintéticas sean una combinación de las originales,

pude lleva a region de decisiones mas pequeñas o especificas, degradando su performance.

Para Adaboost con Random Forest, al ser un clasificador basado en arboles estos son robustos a datos con ruido, y sensibles a la información que se agrega con los datos balanceado, esperando un mejor performace.

Conjunto de Test

Para esta parte se brindo un conjunto de test (RP14_letter_test_parte1.arff), con el cual se comprobaron los clasificadores hallados,obtenido los siguientes resultados

	Parametros	Sin Balanceo	Resample	SMOTE
KNN	infoGain	96.3462	93.9231	96.9615
SVM	C=6 gamma=1	95.0385	94.5769	95.5385
Adaboost Random Forest	Resampling	98.5769	96.9231	98.9231

Conclusiones

Dada las pruebas realizadas con los archivos de entrenamiento, los resultados obtenidos con los archivo de test se encuentran dentro del rango esperado. Y por ensima del umbral preestablecido.
Para el caso de balanceo de resample vemos que como se utiliza muestreo con reposición los algoritmos de clasificación utilizados tienden a sobreentrenar con los datos de entrenamiento, disminuyendo la performance de los clasificadores al utilizar los datos de test.
Para el caso de balanceo con SMOTE se observa una mejora en el conjunto de test, ya que la información que se obtiene al balancear los datos de entrenamiento genera mayor peso, que el ruido que genera introducir estas muestras sintéticas.
En todos los casos el algoritmo de mejor performace fue Adaboot con Random Forest, ya que este es menos sensible a los ruidos introducidos por los algoritmos de balanceo.

Texto en español

Para agregar la información de que el texto se encuentra en idioma español nos planteamos dos opciones, la primera utilizar un clasificador sensible a costos y la segunda realizar un preprocesamiento para variar la ocurrencia relativa de las clases. Viendo los resultados obtenidos en[7] donde marca que en general el primer método logra resultados mejores nos definimos por este.

Visto lo anterior se procedió a modificar la matriz de costos aplicando el meta clasificador CostSensitiveClassifier.

Este meta clasificador hace a su método base sensible al costo. Se pueden utilizar dos métodos para la introducción de la sensibilidad a los costos: ponderar las instancias de entrenamiento de acuerdo al costo total asignado a cada clase, o predecir la clase con el menor costo de error de clasificación esperado.
Antes de adentrarnos en el algoritmo veamos como se define el riesgo condicional que sera clave para la implementación de este:

Dado un conjunto de ejemplo ”x”, la probabilidad de cada clase “j” viene dada por $P(j|x)$, la predicción óptima de Bayes para “x” es la clase “i” que minimiza el riesgo condicional:

$R(i|x)=\sum_{j} P(j|x)C(i,j)$ siendo: $C(i,j)$ el costo de predecir que un ejemplo pertenece a la clase 'i' cuando en realidad pertenece a la clase 'j'

Ecuacion 1

El riesgo condicional $R(i|x)$ es el costo esperado de predecir que x pertenece a la clase “i”. La predicción óptima de Bayes nos garantiza que se alcance el costo total mas bajo posible.

Ahora entraremos en un breve resumen del funcionamiento del algoritmo, por mas detalle referirse a [7].
El algoritmo funciona mediante la creación de múltiples replicas de los datos de entrenamiento mediante bootstrap y aprendiendo el clasificador para cada uno de ellas, se estima la probabilidad de cada clase para cada ejemplo como la fracción de votos que recibe del conjunto. Mediante la “ecuacion 1” se re-etiquetan cada ejemplo de entrenamiento con la clase óptima estimada y se aplica el clasificador para el set de entrenamiento re-etiquetado.

Para el llenado de la matriz de costos se decidió que el costo de clasificar una letra de forma errada debería ser proporcional a la frecuencia de las letras en el idioma español [8].
Para la letra “n” se tomo como la frecuencia la suma de su frecuencia mas la frecuencia de la letra “ñ”.Obteniéndose la siguiente matriz de costos:

0	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88
0.1	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.34	0.34	0	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34
0.4	0.4	0.4	0	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
0.94	0.94	0.94	0.94	0	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94
0.06	0.06	0.06	0.06	0.06	0	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
0.1	0.1	0.1	0.1	0.1	0.1	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.05	0.05	0.05	0.05	0.05	0.05	0.05	0	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51
0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41
0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52
0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62
0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48
0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0	0.55	0.55	0.55	0.55	0.55	0.55	0.55
0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0	0.33	0.33	0.33	0.33	0.33	0.33
0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0	0.47	0.47	0.47	0.47	0.47
0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0	0.07	0.07	0.07	0.07
0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0	0.01	0.01	0.01
0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0	0.02	0.02
0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0	0.07
0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0

Tabla 26: Matriz de Costos- Frecuencia de letras en español

A modo de ejemplo se puede observar que el costo de clasificar incorrectamente una letra “a” es 88 veces mayor que el de clasificar de forma incorrecta una “k”.
De esta forma se logra que la relacional entre los costos sea representativa de la frecuencia de uso en el idioma español.

Resultados

Al correr el conjunto de test contra los algoritmo de clasificación seleccionados se obtiene los siguientes resultados.

	Adaboost-Random Forest			KNN			SVM		
Balanceo	**Sin**	**Resample**	**Smote**	**Sin**	**Resample**	**Smote**	**Sin**	**Resample**	**Smote**
Sin costos	97.52	96.96	97.86	94.58	94.12	96.68	98.42	97.76	95.06
Matriz costos	97.84	97.34	97.92	96.86	95.42	97.24	98.56	98.02	96.74

Tabla 27: Clasificación de datos test

Se realizo una tabla comparativa entre los 3 métodos seleccionados, sin balanceo y con los distintos métodos de balanceo. Para KNN se utilizo el selector de características InfoGain, para SVM los parámetros optimizados C=6 y gamma =1, y el meta-clasificador adaboost con clasificador base Random forest para mejorar su performance.

Conclusiones

Se puede observar como la matriz de costos permitió para todos los casos una mejora de la performance, llegándose por ejemplo para KNN con los datos de trainning sin pre-procesamiento a una mejora superior al 2% en la exactitud.

Estos resultados coinciden con lo esperado ya que el texto de test brindado al estar en español y ser relativamente largo, presenta una frecuencia de aparición de las letras similar a la vista en [7].

El uso de un método sensible a los costos nos permitió incorporar la información de la frecuencia de las letras para el idioma español, para esto se armo la matriz de costos de forma que el costo de clasificar una letra de forma incorrecta sea proporcional a la frecuencia de esta letra en el idioma español.
Por ejemplo para el caso de random forest el peso de la clase se incorpora en dos lugares a la hora de determinar si se debe separar en nuevas ramas (criterio de Gini) y al momento de predecir la clase en los nodos terminales donde se determina mediante “weighted majority vote”.
En el caso de SVM el hiperplano que maximiza la separación entre las clases puede ser llevado hacia las clases mayoritarias logrando así que las instancias que podrían ser dudosas (muy cercanas al hiperplano original) se clasifiquen como la clase minoritaria. En este caso se utilizo el meta clasificador costSensitiveClassifier para hacer referencia a la frecuencia de las letras e ingresarlo

como conocimiento a priori del problema.

En el caso de KNN se realiza un re-weitening de la distancia llevando al método a ser sensible a los costos.
En los tres casos los resultados obtenidos con el metaclasificador sensible a los costos son buenos, presentándose este método como una buena alternativa a la hora de tratar con problemas con clases desbalanceadas. Pudiendo presentar en algunos casos resultados que superan los logrados con under y over-sampling.

Referencia

[1]- Geoff Webb: Decision Tree Grafting From the All-Tests-But-One Partition. In: , San Francisco, CA, 1999
http://ijcai.org/Past%20Proceedings/IJCAI-99%20VOL-2/PDF/007.pdf

[2]- Leo Breiman (2001). Random Forests. Machine Learning. 45(1):5-32.
http://link.springer.com/article/10.1023/A:1010933404324#page-2

[3]- Yoav Freund, Robert E. Schapire: Experiments with a new boosting algorithm. In: Thirteenth International Conference on Machine Learning, San Francisco, 148-156, 1996.
http://machine-learning.martinsewell.com/ensembles/boosting/
http://machine-learning.martinsewell.com/ensembles/boosting/FreundSchapire1996.pdf

[4]- http://weka.sourceforge.net/doc.packages/gridSearch/weka/classifiers/meta/GridSearch.html

[5]- http://weka.wikispaces.com/

[6]- http://www.sandia.gov/~wpk/pubs/publications/smote.pdf

[7]- Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. 1999
[8]- http://www.kriptopolis.com/criptografia-clasica-ii