XGBoost: Algoritmo de Gradiente Extremo Boosting para construir un modelo de aprendizaje automático

Tarea de predicción

Antecedentes

Con la llegada de dispositivos portátiles como Fitbit, recopilar grandes cantidades de datos de actividad personal se ha vuelto más accesible y asequible. Estos dispositivos son adoptados por el movimiento del 'yo' cuantificado, una comunidad de personas que monitorean y miden constantemente diversos aspectos de su vida para mejorar su bienestar, identificar patrones de comportamiento o simplemente por su pasión por la tecnología. Si bien las personas suelen cuantificar la cantidad de una actividad específica que realizan, rara vez evalúan la calidad de su rendimiento en dicha actividad.

En esta práctica en particular, el objetivo es utilizar los datos capturados por acelerómetros colocados en diversas partes del cuerpo (cinturón, antebrazo, brazo y mancuerna) de seis participantes. Se les pidió que realizaran levantamientos de pesas de forma correcta e incorrecta en cinco variaciones diferentes. Para más detalles e información adicional sobre el conjunto de datos, puedes consultar el sitio web http://groupware.les.inf.puc-rio.br/har, específicamente la sección dedicada al conjunto de datos de ejercicios de levantamiento de pesas.

Preparación de los datos

Cargar paquetes, configurar el almacenamiento en caché

```
> library(caret)
> library(corrplot)
> library(Rtsne)
> library(xgboost)
> library(stats)
> library(ggplot2)
> library(knitr)
> knitr::opts_chunk$set(cache=TRUE)
```

Para garantizar un entrenamiento del modelo rápido y preciso, optamos por XGBoost, una implementación del algoritmo de potenciación (*boosting*) de gradiente extremo basado en árboles. XGBoost es conocido por su eficiencia y eficacia en el manejo de grandes conjuntos

de datos, así como por su capacidad para capturar relaciones y patrones complejos dentro de los datos. Al utilizar XGBoost, buscamos optimizar el proceso de entrenamiento y lograr un rendimiento óptimo del modelo.

Obtención de datos

```
> train = read.csv("pml-training.csv")
> test = read.csv("pml-testing.csv")
> dim(train)
[1] 19622   160
> dim(test)
[1]  20 160
```

Los datos de entrenamiento sin procesar constan de 19,622 filas de observaciones y 158 características (predictores). Una de las columnas, denominada "X", representa el número de fila y no se puede utilizar para entrenar el modelo. Además, los datos de prueba constan de 20 filas con las mismas 158 características. El resultado objetivo se indica en una sola columna denominada "classe". Esta columna contiene el resultado o la clase que el modelo pretende predecir.

Limpieza de datos

Para preprocesar los datos de entrenamiento, el primer paso consiste en extraer el resultado objetivo, que representa la calidad de la actividad. Esto nos permite crear un conjunto de datos de entrenamiento modificado que contiene únicamente las variables predictoras, específicamente los monitores de actividad.

```
> outcome.org <- train[["classe"]]
> outcome <- as.factor(outcome.org)
> # Verificar los niveles de la variable de resultado
> levels(outcome)
[1] "A" "B" "C" "D" "E"
```

El resultado tiene 5 niveles en formato de caracteres. Convierte el resultado a numérico, ya que el XGBoost gradient booster solo reconoce datos numéricos.

```
> num.class = length(levels(outcome))
> levels(outcome) = 1:num.class
> head(outcome)
[1] 1 1 1 1 1 1
Levels: 1 2 3 4 5
```

El resultado se elimina de los datos de entrenamiento.

```
> train$classe = NULL
```

Los datos de los acelerómetros en el cinturón, el antebrazo, el brazo y la mancuerna son las características que se extraen con base en estas palabras clave.

```
> # columnas de filtro por: cinturón, antebrazo, brazo, mancuerna
> filter = grepl("belt|arm|dumbell", names(train))
> train = train[, filter]
> test = test[, filter]
```

En lugar de una imputación menos precisa de los datos faltantes, elimina todas las columnas con valores NA.

```
> # eliminar columnas con NA, usar datos de prueba como referencia para NA
> cols.without.na = colSums(is.na(test)) == 0
> train = train[, cols.without.na]
> test = test[, cols.without.na]
```

Preprocesamiento

Comprobar la varianza de las características

En el contexto del análisis de componentes principales (PCA), maximizar la varianza de las características es crucial para lograr la máxima distinción. Esto garantiza que cada característica esté lo más alejada posible (o sea lo más ortogonal posible) de las demás. Al maximizar la varianza, el PCA puede capturar eficazmente los patrones y la variabilidad más significativos dentro de los datos, lo que genera componentes más informativos y representativos.

```
> # comprobar si hay varianza cero
> zero.var = nearZeroVar(train, saveMetrics=TRUE)
> zero.var
```

Ver los resultados en la siguiente página.

No existen características sin variabilidad (todas tienen suficiente varianza). Por lo tanto, no hay ninguna característica que pueda eliminarse.

Gráfico de la relación entre las características y el resultado

	freqRatio	percentUnique	zeroVar	nzv
roll_belt	1.101904	6.7781062	FALSE	FALSE
pitch_belt	1.036082	9.3772296	FALSE	FALSE
yaw_belt	1.058480	9.9734991	FALSE	FALSE
total_accel_belt	1.063160	0.1477933	FALSE	FALSE
gyros_belt_x	1.058651	0.7134849	FALSE	FALSE
gyros_belt_y	1.144000	0.3516461	FALSE	FALSE
gyros_belt_z	1.066214	0.8612782	FALSE	FALSE
accel_belt_x	1.055412	0.8357966	FALSE	FALSE
accel_belt_y	1.113725	0.7287738	FALSE	FALSE
accel_belt_z	1.078767	1.5237998	FALSE	FALSE
magnet_belt_x	1.090141	1.6664968	FALSE	FALSE
magnet_belt_y	1.099688	1.5187035	FALSE	FALSE
magnet_belt_z	1.006369	2.3290184	FALSE	FALSE
roll_arm	52.338462	13.5256345	FALSE	FALSE
pitch_arm	87.256410	15.7323412	FALSE	FALSE
yaw_arm	33.029126	14.6570176	FALSE	FALSE
total_accel_arm	1.024526	0.3363572	FALSE	FALSE
gyros_arm_x	1.015504	3.2769341	FALSE	FALSE
gyros_arm_y	1.454369	1.9162165	FALSE	FALSE
gyros_arm_z	1.110687	1.2638875	FALSE	FALSE
accel_arm_x	1.017341	3.9598410	FALSE	FALSE
accel_arm_y	1.140187	2.7367241	FALSE	FALSE
accel_arm_z	1.128000	4.0362858	FALSE	FALSE
magnet_arm_x	1.000000	6.8239731	FALSE	FALSE
magnet_arm_y	1.056818	4.4439914	FALSE	FALSE
magnet_arm_z	1.036364	6.4468454	FALSE	FALSE
roll_forearm	11.589286	11.0895933	FALSE	FALSE
pitch_forearm	65.983051	14.8557741	FALSE	FALSE
yaw_forearm	15.322835	10.1467740	FALSE	FALSE
total_accel_forearm	1.128928	0.3567424	FALSE	FALSE
gyros_forearm_x	1.059273	1.5187035	FALSE	FALSE
gyros_forearm_y	1.036554	3.7763735	FALSE	FALSE
gyros_forearm_z	1.122917	1.5645704	FALSE	FALSE
accel_forearm_x	1.126437	4.0464784	FALSE	FALSE
accel_forearm_y	1.059406	5.1116094	FALSE	FALSE
accel_forearm_z	1.006250	2.9558659	FALSE	FALSE
magnet_forearm_x	1.012346	7.7667924	FALSE	FALSE
magnet_forearm_y	1.246914	9.5403119	FALSE	FALSE
magnet_forearm_z	1.000000	8.5771073	FALSE	FALSE

Graficar la relación entre las características y el resultado. En el gráfico a continuación, cada característica tiene una distribución relativamente uniforme entre los 5 niveles de resultado (A, B, C, D, E).

> featurePlot(train, outcome, "strip")

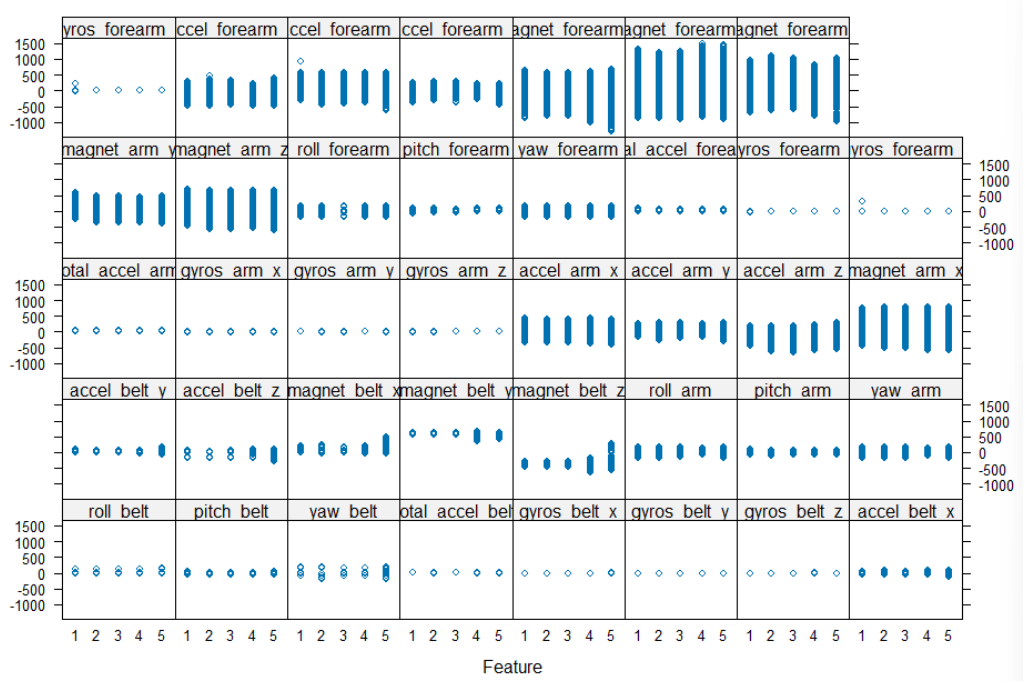
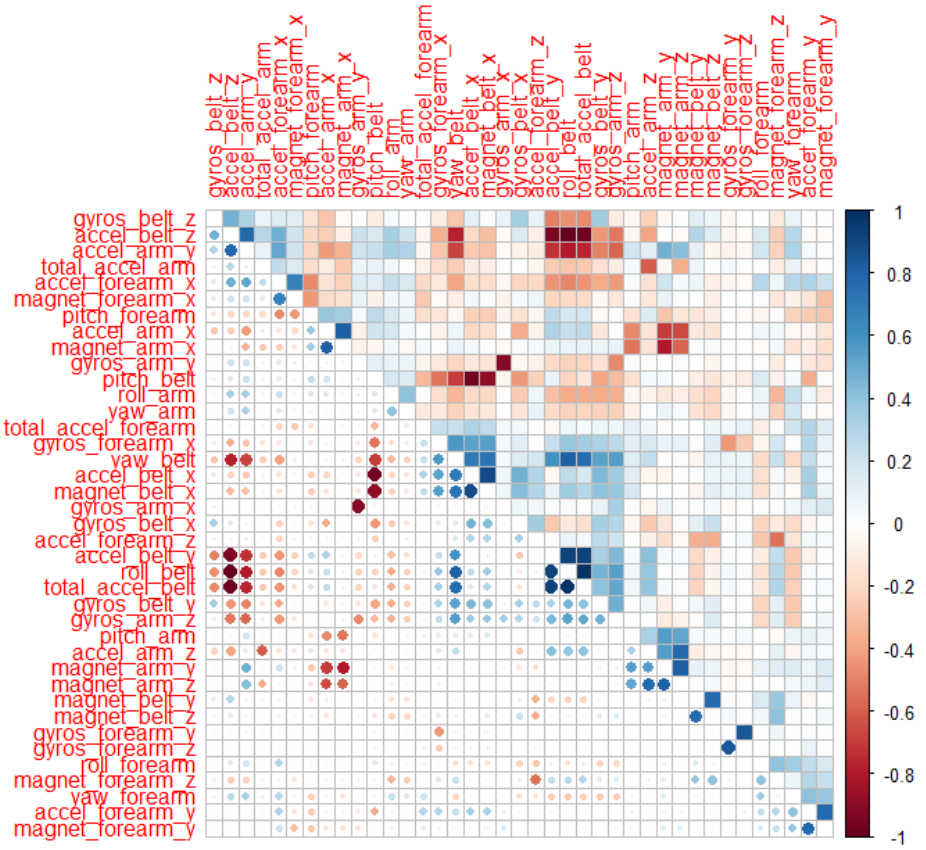


Gráfico de la matriz de correlación

Graficaremos una matriz de correlación entre las características.
Al considerar un conjunto de características, es deseable que presenten una baja correlación, lo que indica ortogonalidad. Al evaluar la correlación promedio mediante un gráfico de la matriz de correlación, podemos determinar el grado de correlación entre las características. En este caso, el gráfico revela que la correlación promedio no es excesivamente alta, lo que sugiere que las características están relativamente poco correlacionadas. Por lo tanto, he decidido no continuar con pasos de preprocesamiento adicionales, como el análisis de componentes principales (PCA).

```
> corrplot.mixed(cor(train), lower="circle", upper="color",
+    tl.pos="lt", diag="n", order="hclust", hclust.method="complete")
```

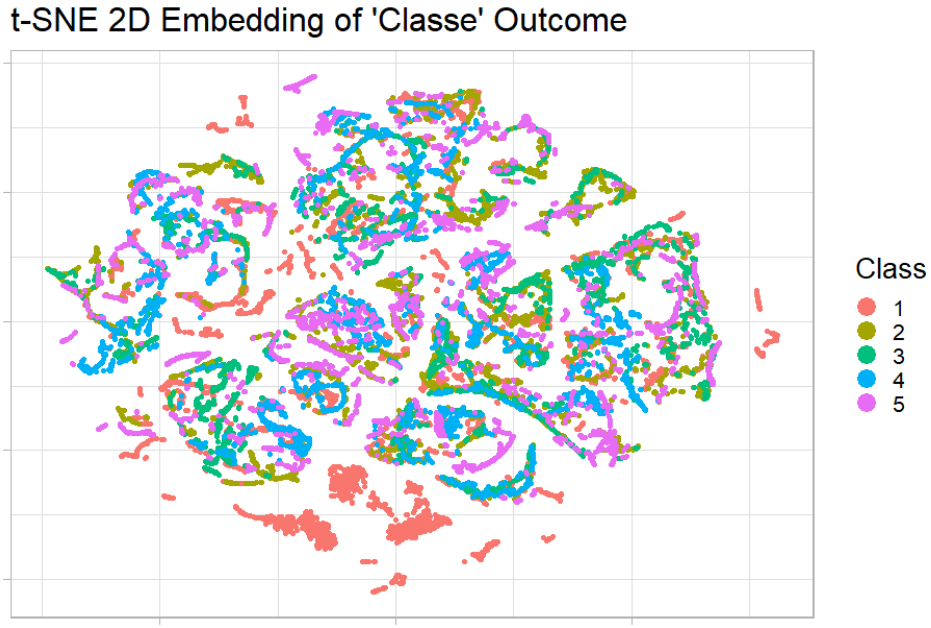


Gráfica tSNE

La visualización t-SNE (Incrustación Estocástica de Vecinos Distribuida t, *t-Distributed Stochastic Neighbor Embedding*) es una técnica que se utiliza para reducir la dimensionalidad de las características multidimensionales y proyectarlas en un plano 2D. Su objetivo es capturar la estructura y las relaciones subyacentes en los datos. En la gráfica t-SNE que se

muestra a continuación, no se observa una separación evidente ni una agrupación clara entre los cinco niveles de la variable de resultado (A, B, C, D, E). Por lo tanto, resulta difícil extraer conclusiones definitivas o construir manualmente una ecuación de regresión debido a la irregularidad y la falta de patrones claros en los datos.

```
> # t-Distributed Stochastic Neighbor Embedding
> tsne = Rtsne(as.matrix(train), check_duplicates=FALSE, pca=TRUE,
+   perplexity=30, theta=0.5, dims=2)
> embedding = as.data.frame(tsne$Y)
> embedding$Class = outcome
> g = ggplot(embedding, aes(x=V1, y=V2, color=Class)) +
+   geom_point(size=1.25) +
+   guides(colour=guide_legend(override.aes=list(size=6))) +
+   xlab("") + ylab("") +
+   ggtitle("t-SNE 2D Embedding of 'Classe' Outcome") +
+   theme_light(base_size=20) +
+   theme(axis.text.x=element_blank(),
+     axis.text.y=element_blank())
> print(g)
```



Construir un modelo de aprendizaje automático

Ahora, construiremos un modelo de aprendizaje automático para predecir la calidad de la actividad (resultado de classe) a partir de los monitores de actividad (características o predictores) mediante el algoritmo de potenciación de gradiente extremo de XGBoost.

Datos de XGBoost

XGBoost solo admite datos matriciales numéricos. Convierte todos los datos de entrenamiento, prueba y resultados a una matriz.

```
> # convert data to matrix
> train.matrix = as.matrix(train)
> mode(train.matrix) = "numeric"
> test.matrix = as.matrix(test)
> mode(test.matrix) = "numeric"
> # convert outcome from factor to numeric matrix
> #   xgboost takes multi-labels in [0, numOfClass)
> y = as.matrix(as.integer(outcome)-1)
```

Parámetros de XGBoost

Establece los parámetros de XGBoost para la validación cruzada y el entrenamiento.
Establece un objetivo de clasificación multiclase como función de aprendizaje de la potenciación de gradiente.
Establece la métrica de evaluación en merror, la tasa de error multiclase.

```
> # xgboost parameters
> param <- list("objective" = "multi:softprob",    # multiclass classification
+   "num_class" = num.class,    # number of classes
+   "eval_metric" = "merror",    # evaluation metric
+   "nthread" = 8,   # number of threads to be used
+   "max_depth" = 16,    # maximum depth of tree
+   "eta" = 0.3,    # step size shrinkage
+   "gamma" = 0,    # minimum loss reduction
+   "subsample" = 1,    # part of data instances to grow tree
+   "colsample_bytree" = 1,  # subsample ratio of columns when constructing each tree
+   "min_child_weight" = 12  # minimum sum of instance weight needed in a child
+ )
```

Tasa de error esperada

La tasa de error esperada es inferior al 1 % para una buena clasificación. Realiza una validación cruzada para estimar la tasa de error mediante una validación cruzada cuádruple, con 200 épocas para alcanzar una tasa de error esperada inferior al 1 %.

Validación cruzada cuádruple (4-fold)

```
> # set random seed, for reproducibility
> set.seed(1234)
> # k-fold cross validation, with timing
> nround.cv = 200
> system.time( bst.cv <- xgb.cv(param=param, data=train.matrix, label=y,
+   nfold=4, nrounds=nround.cv, prediction=TRUE, verbose=FALSE) )
   user  system elapsed
  62.63    2.50   67.20
```

El tiempo transcurrido es de aproximadamente 63 segundos (1 minutos y unos segundos).

```
> tail(bst.cv$evaluation_log)
```

	iter	train_merror_mean	train_merror_std	test_merror_mean	test_merror_std
	<num>	<num>	<num>	<num>	<num>
1:	195	0	0	0.006370624	0.001555904
2:	196	0	0	0.006217719	0.001516092
3:	197	0	0	0.006217719	0.001516092
4:	198	0	0	0.006268698	0.001528951
5:	199	0	0	0.006268698	0.001528951
6:	200	0	0	0.006166782	0.001494608

```
> # index of minimum merror
> evaluation_df <- as.data.frame(bst.cv$evaluation_log)
> min.merror.idx <- which.min(evaluation_df$test_merror_mean)
> min.merror.idx
[1] 184
> bst.cv$evaluation_log[min.merror.idx,]
```

	iter	train_merror_mean	train_merror_std	test_merror_mean	test_merror_std
	<num>	<num>	<num>	<num>	<num>
1:	184	0	0	0.006166782	0.001494608

La mejor tasa de error mínima de la validación cruzada, test.merror.mean, es de aproximadamente 00061 (0.6 %), y se produjo en la iteración 184.

Matriz de confusión

Tabula las predicciones del modelo de validación cruzada frente a las verdaderas.

```
> # get CV's prediction decoding
> pred.cv = matrix(bst.cv$pred, nrow=length(bst.cv$pred)/num.class, ncol=num.class)
> pred.cv = max.col(pred.cv, "last")
> # confusion matrix
> confusionMatrix(factor(y+1), factor(pred.cv))
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5
         1 5567   10    2    1    0
         2   15 3761   21    0    0
         3    2   18 3388   14    0
         4    0    0   22 3190    4
         5    0    1    0   11 3595

Overall Statistics

               Accuracy : 0.9938
                 95% CI : (0.9926, 0.9949)
    No Information Rate : 0.2846
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9922

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
Sensitivity            0.9970   0.9923   0.9869   0.9919   0.9989
Specificity            0.9991   0.9977   0.9979   0.9984   0.9993
Pos Pred Value         0.9977   0.9905   0.9901   0.9919   0.9967
Neg Pred Value         0.9988   0.9982   0.9972   0.9984   0.9998
Prevalence             0.2846   0.1932   0.1750   0.1639   0.1834
Detection Rate         0.2837   0.1917   0.1727   0.1626   0.1832
Detection Prevalence   0.2844   0.1935   0.1744   0.1639   0.1838
Balanced Accuracy      0.9980   0.9950   0.9924   0.9952   0.9991
```

La matriz de confusión muestra que la concentración de predicciones correctas está en la diagonal, como se esperaba.

La precisión promedio es del 99.38 %, con una tasa de error del 0.62 %. Por lo tanto, se cumple una tasa de error esperada inferior al 1 %.

Entrenamiento del modelo

Ajusta el modelo de gradient boosting XGBoost a todos los datos de entrenamiento.

```
> system.time( bst <- xgboost(param=param, data=train.matrix, label=y,
+   nrounds=min.merror.idx, verbose=0) )
   user  system elapsed
  16.22    0.64   23.10
```

El tiempo transcurrido es de aproximadamente 16 segundos.

Predicción de los datos de prueba

```
> # xgboost predict test data using the trained model
> pred <- predict(bst, test.matrix)
```

```
> head(pred, 10)
 [1] 7.850143e-04 9.969912e-01 1.699796e-03 1.538672e-04 3.701838e-04 9.991676e-01
5.930709e-04 2.202196e-04
 [9] 5.147830e-06 1.402383e-05
```

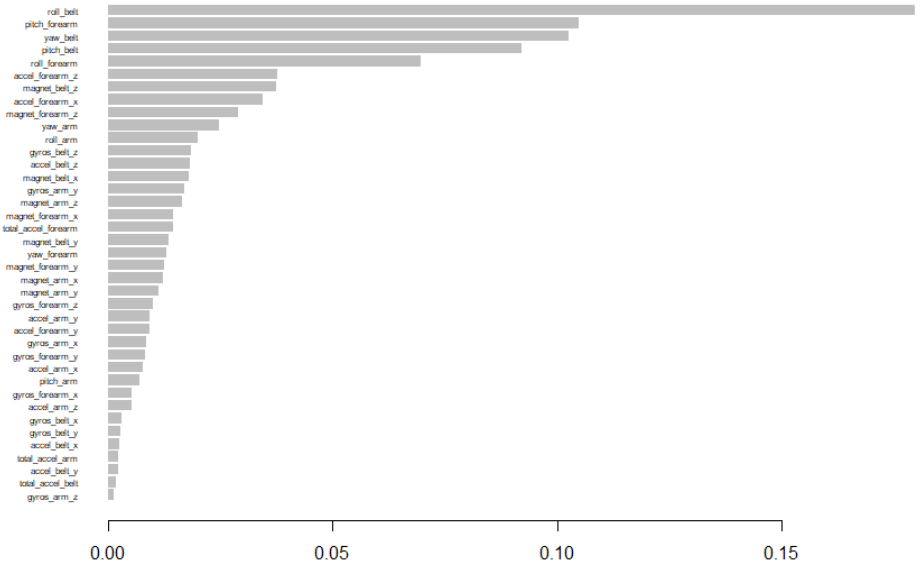
Posprocesamiento

El resultado de la predicción es la probabilidad predicha de los 5 niveles (columnas) del resultado.
Descifra los 5 niveles cuantitativos de resultados en letras cualitativas (A, B, C, D, E).

```
> # decode prediction
> pred = matrix(pred, nrow=num.class, ncol=length(pred)/num.class)
> pred = t(pred)
> pred = max.col(pred, "last")
> pred.char = toupper(letters[pred])
```

Importancia de las características

```
> # get the trained model
> model = xgb.dump(bst, with_stats=TRUE)
> # get the feature real names
> names = dimnames(train.matrix)[[2]]
> # compute feature importance matrix
> importance_matrix = xgb.importance(names, model=bst)
> # plot
> gp = xgb.plot.importance(importance_matrix)
```



> print(gp)

	Feature	Gain	Cover	Frequency	Importance
	<char>	<num>	<num>	<num>	<num>
1:	roll_belt	0.179693466	0.121764440	0.057974580	0.179693466
2:	pitch_forearm	0.104947326	0.092465767	0.059614596	0.104947326
3:	yaw_belt	0.102475208	0.095275090	0.091266913	0.102475208
4:	pitch_belt	0.092135521	0.071272885	0.067650677	0.092135521
5:	roll_forearm	0.069670769	0.064869754	0.057236572	0.069670769
6:	accel_forearm_z	0.037746486	0.026622591	0.033128331	0.037746486
7:	magnet_belt_z	0.037438375	0.039328015	0.027634276	0.037438375
8:	accel_forearm_x	0.034442852	0.026717664	0.021238212	0.034442852
9:	magnet_forearm_z	0.029086592	0.025872505	0.031570316	0.029086592
10:	yaw_arm	0.024875200	0.020199864	0.022222222	0.024875200
11:	roll_arm	0.020061557	0.022156598	0.037884379	0.020061557
12:	gyros_belt_z	0.018613803	0.037328584	0.018696187	0.018613803
13:	accel_belt_z	0.018396163	0.021744467	0.025256253	0.018396163
14:	magnet_belt_x	0.018055847	0.021371456	0.026486265	0.018055847
15:	gyros_arm_y	0.017025835	0.024099412	0.024846248	0.017025835
16:	magnet_arm_z	0.016503448	0.025150016	0.024272243	0.016503448
17:	magnet_forearm_x	0.014500613	0.017758645	0.022140221	0.014500613
18:	total_accel_forearm	0.014454438	0.011082587	0.010332103	0.014454438
19:	magnet_belt_y	0.013472704	0.019088688	0.018204182	0.013472704
20:	yaw_forearm	0.012920827	0.011525266	0.023124231	0.012920827
21:	magnet_forearm_y	0.012472699	0.016735308	0.021894219	0.012472699
22:	magnet_arm_x	0.012335533	0.006822218	0.011890119	0.012335533
23:	magnet_arm_y	0.011368374	0.019503077	0.021074211	0.011368374
24:	gyros_forearm_z	0.010087970	0.012498078	0.014596146	0.010087970
25:	accel_arm_y	0.009375663	0.012738708	0.017466175	0.009375663
26:	accel_forearm_y	0.009330366	0.012748419	0.022140221	0.009330366
27:	gyros_arm_x	0.008447371	0.015855702	0.024518245	0.008447371
28:	gyros_forearm_y	0.008301205	0.013944945	0.025830258	0.008301205
29:	accel_arm_x	0.007769142	0.016177037	0.020500205	0.007769142
30:	pitch_arm	0.007106283	0.012049470	0.022878229	0.007106283
31:	gyros_forearm_x	0.005417409	0.009446135	0.016072161	0.005417409
32:	accel_arm_z	0.005278959	0.008382962	0.019188192	0.005278959
33:	gyros_belt_x	0.002989696	0.009767955	0.016810168	0.002989696
34:	gyros_belt_y	0.002935031	0.013814657	0.007462075	0.002935031
35:	accel_belt_x	0.002510063	0.008031963	0.010496105	0.002510063
36:	total_accel_arm	0.002334722	0.003809815	0.008938089	0.002334722
37:	accel_belt_y	0.002290135	0.005811711	0.004920049	0.002290135
38:	total_accel_belt	0.001786903	0.002294077	0.003116031	0.001786903
39:	gyros_arm_z	0.001345448	0.003873466	0.009430094	0.001345448
	Feature	Gain	Cover	Frequency	Importance

El gráfico de importancia de las características es útil para seleccionar solo las mejores características con la mayor correlación con el/los resultado(s). Para mejorar el rendimiento del ajuste del modelo (tiempo o sobreajuste), se pueden eliminar las características menos importantes.

Resumen

En esta práctica, se implementó un algoritmo XGBoost para la clasificación multiclase. Los datos de entrenamiento se utilizaron para entrenar el modelo y se realizó una validación cruzada para evaluar su rendimiento. El modelo entrenado se aplicó posteriormente a los datos de prueba para realizar predicciones. Además, se calculó y visualizó la importancia de las características para comprender la importancia de las diferentes características en la tarea de clasificación.