****

**Escuela Superior Politécnica del Litoral.**

**Mineria de datos.**

**Reporte final del proyecto.**

**Prof.** Carmen Vaca PhD.

**Paralelo 1.**

**Integrantes**:

* Javier de Jesús Anchundia Rosado.
* Miguel Angel Patiño San-yeng.
* Bryan Cesar Tutiven Acosta.

1. **Introducción**.

Un desafío central para Spotify es recomendar la música adecuada a cada usuario. Si bien existe una gran cantidad de trabajo relacionado en los sistemas de recomendación, hay muy poco trabajo o datos que describen cómo los usuarios interactúan secuencialmente con el contenido transmitido que se les presenta. En particular dentro de la música, la cuestión de si, y cuándo, un usuario omite una pista, es una señal de retroalimentación implícita importante.

Dicho lo anterior se plantea el siguiente objetivo: Crear un modelo predictivo que determine si los usuarios de Spotify van a omitir o escuchar la canción actual en su sesión de streaming (ya sea que forme parte de su lista de reproducción, en una reproducción aleatoria o en cualquier otro contexto dentro de las opciones de spotify).

1. **Descripción del dataset.**

Se cuenta con dos conjuntos de datos:

1. El primero contiene información de las sesiones asociadas a la interacción de los usuarios con las canciones. No hay ningún tipo de identificador de usuario, de ahí el desafío ya que. Son un total de 130 millones de sesiones, cada sesión cuenta con máximo 20 canciones y los campos son los siguientes:

**Session\_id**: identificador de la sesión.

**Session\_position**: numero ordenado {1-20}, determina la posición de la canción en la sesión.

**Track\_id\_clean:**  identificador de la canción reproducida.

**Skip\_1**: booleano, indica si la canción fue reproducida muy brevemente.

**Skip\_2**: booleano, indica si la canción fue reproducida no tan breve.

**Skip\_3**: booleano, indica si gran parte de la canción fué reproducida.

**Not\_skipped**: booleano, indica si la canción fué reproducida en su totalidad.

**No\_pause\_before\_play**: booleano, indica si no hubo pausa antes de dar play.

**Long\_pause\_before\_play**: booleano, indica si hubo una larga pausa antes de dar play.

**Short\_pause\_before\_play**: booleano, indica si hubo una corta pausa antes de dar play.

**Hist\_user\_behavior\_n\_seekfwd:** número indicando las veces que el usuario adelantó la canción.

**Hist\_user\_behavior\_n\_seekback:** número indicando las veces que el usuario retrocedió la canción.

**Hour\_of\_day**: número entre {0-23} que representa la hora del dia.

**Date**: E.g. 2018-09-18.

**Premium**: booleano, indica si es usuario premium.

1. El segundo dataset contiene información de las canciones, como el id, popularidad y |vectores de acústica. Además contiene otros 16 features respecto a la característica de la acústica de las canciones, como beat strength, bounciness, danceability, mean dynamic range, energy, flatness, instrumentalness, liveness, loudness, mechanism, tempo, organism, speechiness and valence. Además 8 features de vectores de acústica. Todos son numéricos y vienen pre-procesados por los servicios de Spotify.

**III. Análisis exploratorio.**

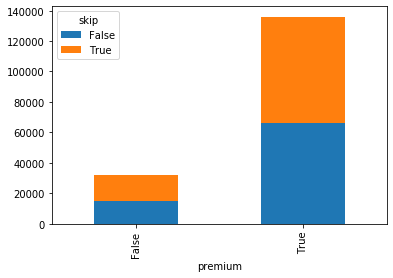
El análisis exploratorio se lo realizó con una versión MINI de los datasets, el cual es una muestra aleatoria con cerca de 8400 sesiones. Cada sesión puede tener hasta 20 canciones.

**Visualización de los datos.**

##### Número de tracks y proporción de skips vs hora del día.

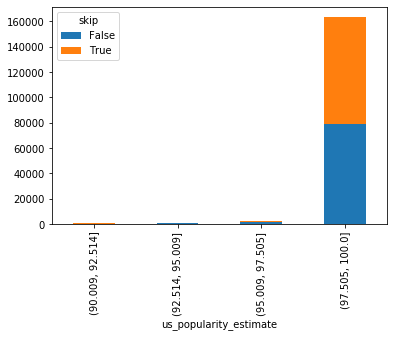
El objetivo de este gráfico es visualizar las proporciones de los tracks del dataset que han formado parte de las sesiones con respecto a la variable output (skip). Se notan similares proporciones tanto para las canciones que han sido saltadas como las que no para todas las horas del dia registradas. Esto ayuda a evitar posibles casos de majority class o ver que tan balanceado están los datos y también muestra la relevancia que tiene la hora del día sobre la cantidad de canciones que son reproducidas.

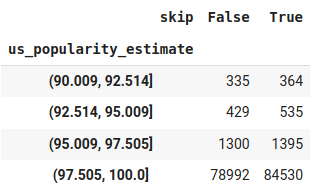
##### Número de tracks y proporción de skips vs si el usuario es premium o no.

****

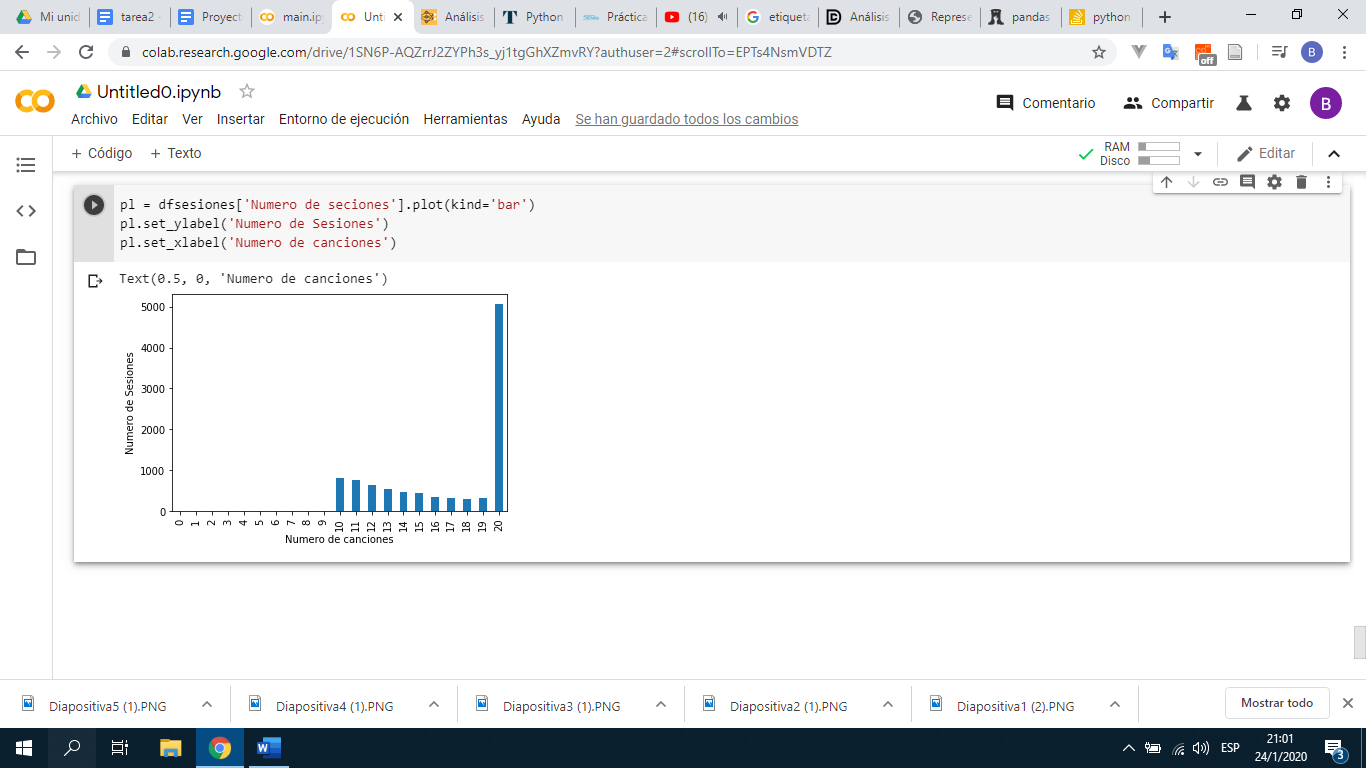
Un factor importante para saber si un usuario hará skip de una canción, es si el usuario es premium o no, debido a las restricciones de spotify para realizar dicha acción para los usuarios no premium. A pesar de ello se muestran similares proporciones para la cantidad de skips en los dos tipos de usuarios.

* Proporción de skips respecto a la popularidad de las canciones en Estados Unidos.





* Total de sesiones vs total de canciones

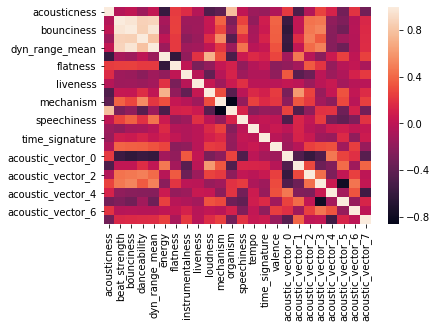
****

Para identificar cuál es la tendencia de los usuarios de quedarse en una playlist, se realizó un gráfico que permita saber cuántas sesiones llegan a un máximo de 20 canciones, en el plot se evidencia que un gran número de sesiones (más de 5000) fueron escuchadas 20 canciones y que es normal que un usuario escuche 10 canciones en adelante por sesión.

**IV. Transformación de datos.**

**Escalado.**

Min max scaler fué usado para escalar algunos features asociados con las sesiones. Los campos fueron: Hist\_user\_behavior\_n\_seekfwd, Hist\_user\_behavior\_n\_seekback y Hour\_of\_day.

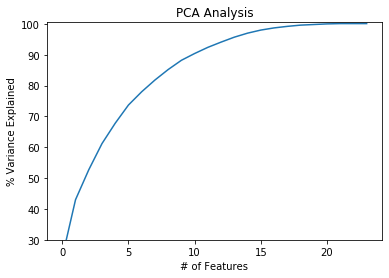


Se normalizaron los datos relacionados con la acústica de los tracks. Campos como el acousticness, beat\_strength, bounciness, danceability, y acoustic vectors.

El mapa de calor de la derecha muestra correlaciones fuertes entre algunas de las variables de la acústica de los tracks, lo cual, podría ser un indicador de redundancia en los datos. Esto podría dar un indicio sobre si es viable o no aplicar PCA

**Reducción de dimensionalidad.**

Se aplicó PCA para reducir la dimensionalidad de los datos relacionados a la acústica de las canciones. Dichos features son un total de 24, los cuales se redujeron a 15. Esto apoyándose en la gráfica del porcentaje de explicación de la varianza y el número de features que se muestra a continuación. Con 15 componentes se logra explicar más del 95% de la varianza.

****

**Union de Datasets**

Para poder predecir si una canción va a ser pasada o no, vamos a usar las características de la canción que se quiere evaluar, y las características tanto de la canción que se escuchó previamente como posteriormente en la misma sesión.

Con el fin de lograr lo anterior, juntamos el dataset consigo de tal forma que en cada fila se tenga información de la canción anterior y actual. Como es de esperarse la primera canción en una sesión no tiene una canción previa, por lo que se decidió que la primera canción se tenga así misma como canción anterior. Esto podría cambiar en un futuro, dependiendo de la eficiencia de los modelos que se obtengan.

**V. Modelos y resultados.**

Se probaron 4 algoritmos:

* Clasificador Naive bayes (GaussianNB de scikit-learn).
* Clasificador Decision Tree (DecisionTreeClassifier de scikit-learn).
* Ensemble: Decision Tree usando Bagging (BaggingClassifier de scikit-learn).
* Ensemble: Gradient Boosting Trees (XGBClassifier de xgboost).
* ## < ----------------- TODO: Clasificador lineal.

La metodología usada para entrenar cada uno de los modelos consistió en variar las configuraciones para cada algoritmo, con el objetivo de elegir la mejor configuración, y hacer la comparación de rendimiento respectiva.

La evaluación de cada configuración en todos los algoritmos se realizó usando cross-validation con 10-fold.

Las configuraciones son las Siguientes

* **GaussianNB**:

Existen 2 hiper-parámetros en este algoritmo

var\_smoothing: Es la porción de la mayor varianza que se toma de todos features que se añaden a la varianzas utilizadas para el cálculo de estabilidad, su valor por defecto es 1xe-09.

Priors: Sirve para poder priorizar una de las clases en el output, si este valor es seteado, entonces ya no se toma en cuenta las probabilidades según la distribución de la data.

* **DecisionTreeClassifier:**

Para este algoritmo se se tomaron en cuenta 4 parámetros

* + max\_depth = np.linspace(1, 32, 32) permitirá definir la profundidad que tendrá del Decision Tree que puede ser desde 1 hasta 32.
  + min\_samples\_splits = np.linspace(0.1, 1.0, 10) define la proporción mínima de samples comenzando desde 0.1, 0.2, y seguirá aumentando hasta que se usen todos.
  + Min\_samples\_leaf = np.linspace(0.1, 0.5, 5) va definiendo proporciones de 0.1, 0.15, 0.20 hasta 0.5 para entrenar el decision tree.
  + max\_features = [10,20,30,40,50] está basado en las cantidades que posee nuestro dataset, ya que nuestra data posee más de 60 features y de los cuales se tomarán como máximo
* **BaggingClassifier**:
  + n\_estimators = [80, 90, 100, 150, 200]. Define el número de estimadores base del ensemble, el cual corresponde al número de subárboles a usar en el modelo.
  + max\_samples = [0.4, 0.6, 0.8, 1.0]. Define las fracción de samples que se tomarán para entrenar cada estimador base.
* **XGBClassifier:** 
  + max\_depths = [10, 11, 12, 13]. Valores para la profundidad máxima de los árboles que son los estimadores base
  + etas = [0.01, 0.05, 0.1, 0.15, 0.2]. Los valores para el learning rate, ya que usa gradient descent.
  + n\_estimators = [90, 100, 110, 120]. Los números de estimadores base del ensemble.

**Evaluación de los modelos variando las configuraciones.**

Se realizó una iteración por cada variante de las configuraciones de los algoritmos, luego con los resultados de usar 10-fold se procedió a elegir la mejor configuración.

La evaluación se la realizó sobre una pequeña muestra aleatoria del dataset, debido al alto tiempo de ejecución que resulta al probar para cada configuración, y los limitados recursos computacionales que poseemos.

1. **GaussianNB.**

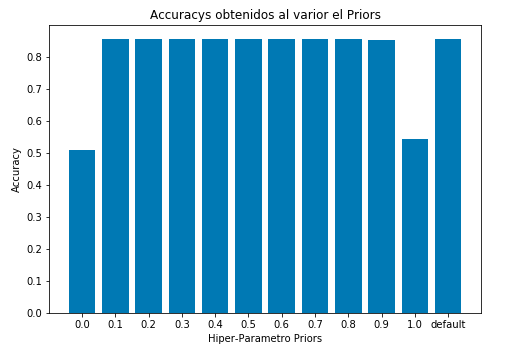
**1.1 Elección de la mejor configuración.**

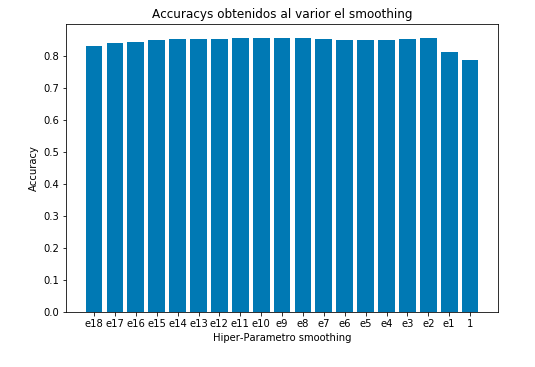
Lo siguiente son gráficos de barras que representan el accuracy obtenido al variar los hiperparametros usando cross-validation, además de variar el dataset al cual se aplica la validación.

*Primer Dataset*

Mejores Hiper-Parámetros:

Prios= default (0.867 de accuracy)

Smoothing= 1xe-09 (0.861 de accuracy) 

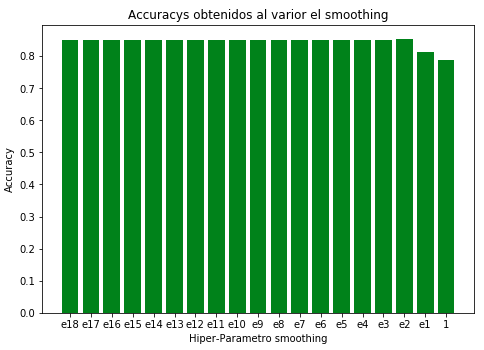
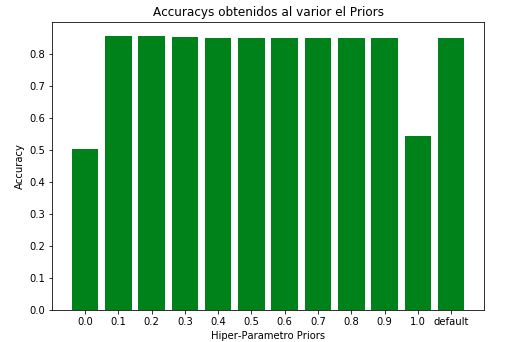


Segundo Dataset

Mejores Hiper-Parámetros:

Prios= [0.1-0.9] (0.857 de accuracy)

Smoothing= 1xe-02 (0.8547 de accuracy)

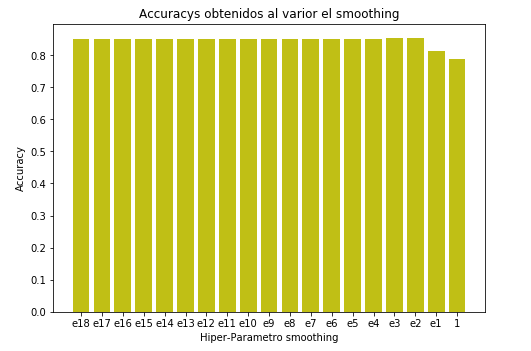
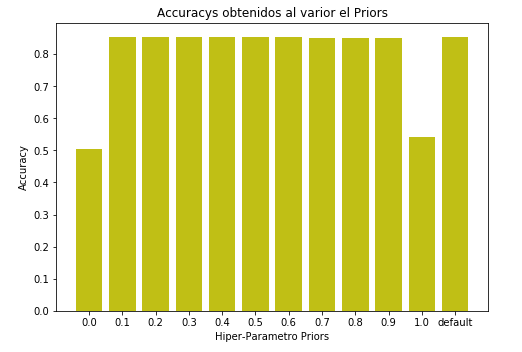


*Tercer Dataset*

Mejores Hiper-Parámetros:

Prios= [0.1-0.9] (0.854 de accuracy)

Smoothing= 1xe-02 (0.854 de accuracy)

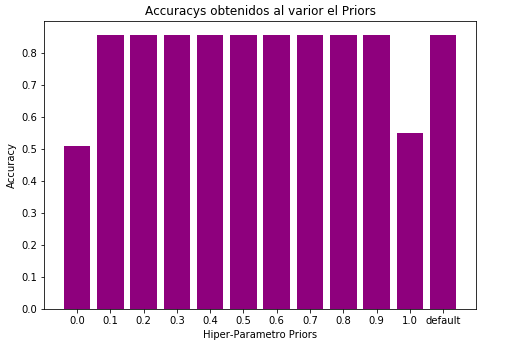
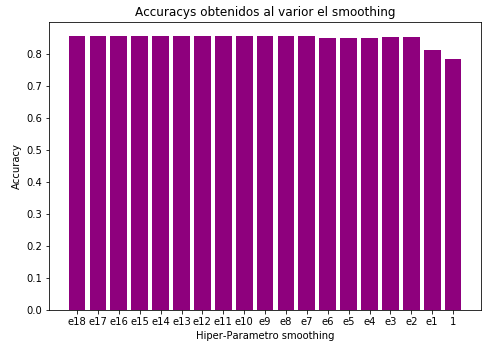


*Cuarto Dataset*

Mejores Hiper-Parámetros:

Prios= [0.9-0.1] (0.8569 de accuracy)

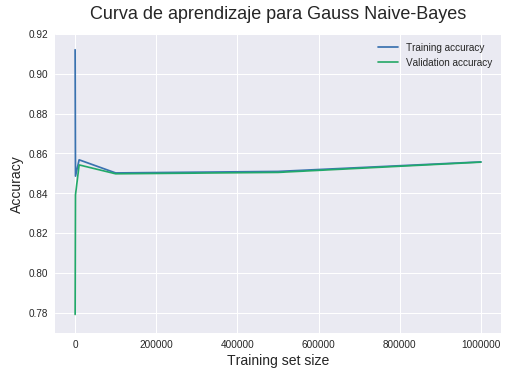
Smoothing= 1xe-08 (0.8569 de accuracy)



Al comparar los resultados de los accuracies de los diferentes datasets, nos podemos dar cuenta de que en realidad la mayoría da un resultado bastante parecido, pero se debe evitar los valores muy extremistas como una proporción de probabilidades de [0.0-1.0] o [1.0-0.0] en el priors o valores muy altos en el smoothing como 1xe-01 o 1xe0.

**1.2 Curvas de aprendizaje y cantidad de datos para un buen modelo.**

Se escogió la configuración de hiper-parámetros que vienen por defecto y se calculó el accuracy del validation y del testing set en otro cross-validation. La gráfica siguiente representa el promedio de accuracy de un cross-validation en función del tamaño del dataset para el training y validation.

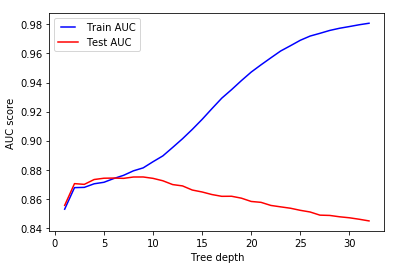


En el gráfico se puede observar como con valores menores que 100 en el dataset existe overfitting al haber un mayor accuracy en el training y un accuracy bajo en el validation. Ambos valores en el accuracy crecen de forma pareja con un dataset de tamaño 10000 y se nota un crecimiento mucho más pronunciado del accuracy con un dataset de tamaño 1000000, por lo que se concluye que se podría obtener un mejor modelo mientras mayor cantidad de datos este posea evitando underfitting, y que el validation y el testing tengan valores similares, denota que no existe overfitting.

1. **DecisionTreeClassifier.**

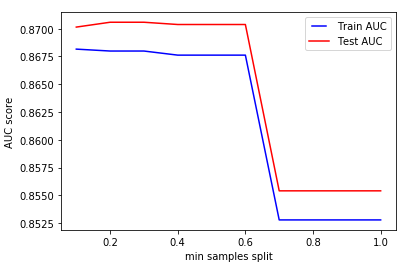
Para elegir los mejores parámetros para construir nuestro modelo final del decision Tree se tomaron en cuenta la mejor configuración en base a los parámetros ya antes de descritos como lo profundidad máxima del árbol, mínimo de de samples para splits y hojas; por último máximo de features. La métrica de de evaluación es AUC que es área bajo la curva de ROC , esta métrica es una buena para evaluar nuestro rendimiento debido a que manejamos un target binario.

2.1 *Parametro para evaluar max\_depth*

**

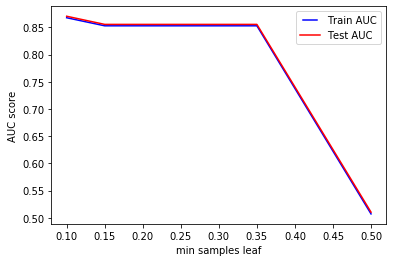
En la gráfica de score AUC vs tamaño del árbol podemos identificar que nuestro modelo se sobreajusta para valores de gran profundidad, el decision tree también predice de manera efectiva los datos del train pero no puede generalizar de manera correcta los nuevos datos.

*2.2 Parametro para evaluar el min samples split*

**

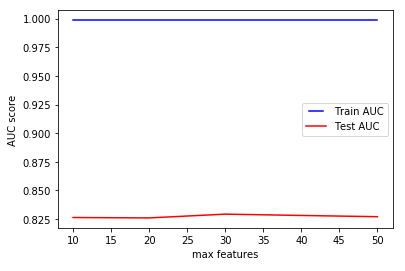
En esta gráfica de AUC score vs min samples split se puede evidenciar que cuando consideramos poco más del 45% de nuestra muestras en cada nodo, el modelo no puede aprender lo suficiente, de manera que se genera un caso de underfitting.

*2.3 Parametro para evaluar min samples leaf*

**

En la gráfica de AUC score vs el mínimo de muestras en la hoja del decision tree, se puede evidenciar que cerca del 45% en adelante el modelo no aprende lo suficiente y que del 15 a 30 % se da una buena predicción de nuevos datos en el modelo.

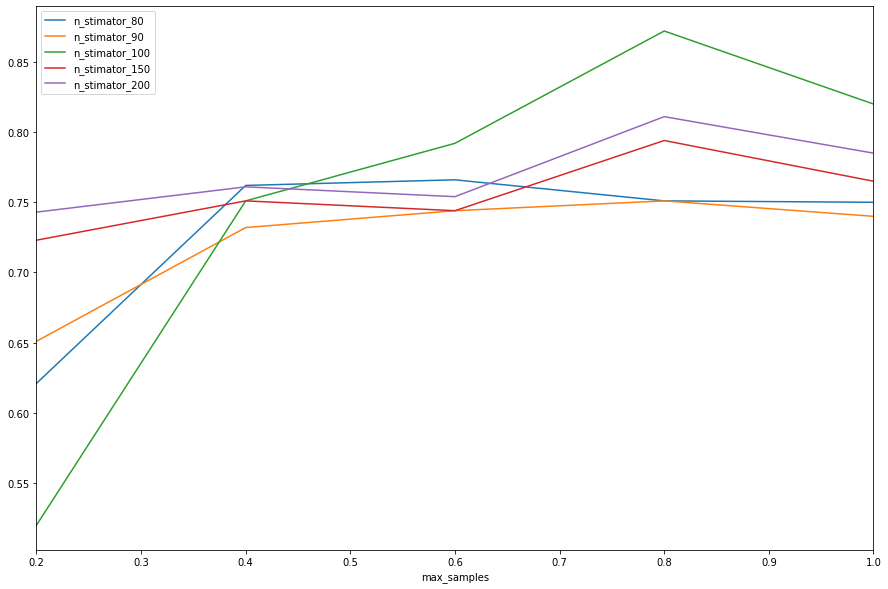
*2.4 Parametro para evaluar max de features*

**

El gráfico de AUC score vs max features representa las características a considerar cuando se busca la mejor división y se puede evidenciar que es existe ovefitting.

1. **BaggingClassifier**.
   1. **Elección de la mejor configuración.**

La siguiente gráfica muestra el resultado del score (acurracy) obtenido para cada configuración usando cross-val-score de scikit-learn

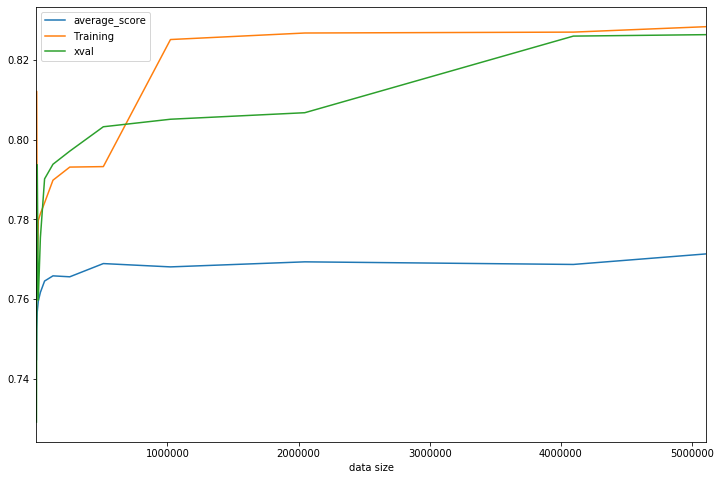
****

Usando este resultado decidimos que la mejor configuración corresponde a: **N\_estimators** = 100 y **Max\_samples** = 0.8, ya que dichas configuraciones obtuvieron un mejor acurracy

* 1. **Curvas de aprendizaje y cantidad de datos para un buen modelo.**

*Score vs número de samples.*

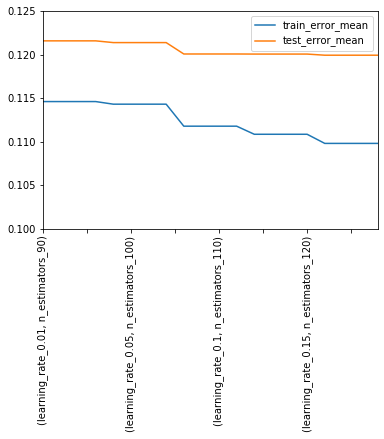
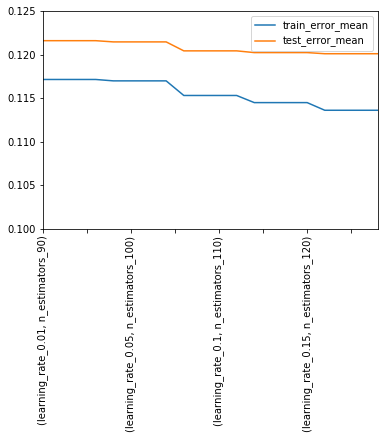
Se muestra la curva de aprendizaje usando la métrica de acurracy para el Training y los valores obtenidos en cross validation con 10-fold.



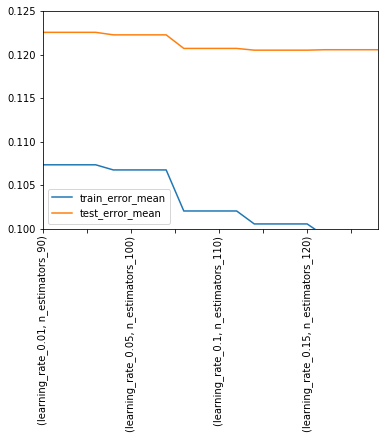
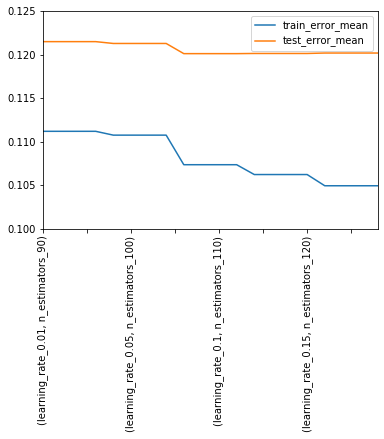
1. **XGBClassifier.**
   1. **Rendimiento del ensemble y elección de la mejor configuración.**

Haciendo variaciones de la profundidad de los estimadores base (max\_depth), el learning rate (eta) y aumentando el tamaño del ensemble (n\_estimator), se se calculó el M.A.E. mean absolute error, y se obtuvieron los siguientes resultados:

*Max\_depth = 10 Max\_depth = 11*



*Max\_depth = 12 Max\_depth = 13*



Las gráficas muestran los errores promedio para diferentes modelos con diferentes configuraciones. Al aumentar la profundidad de los árboles y aumentar el tamaño del ensemble (n\_estimators) las curvas de train y test se separan poco, lo cual es una señal de overfitting. Entonces se procede a elegir la configuración que reduzca la distancia entre los errores de train y test, y reduzca el error en general.

Usando este resultado decidimos que la mejor configuración corresponde a:

**N\_estimators** = 110

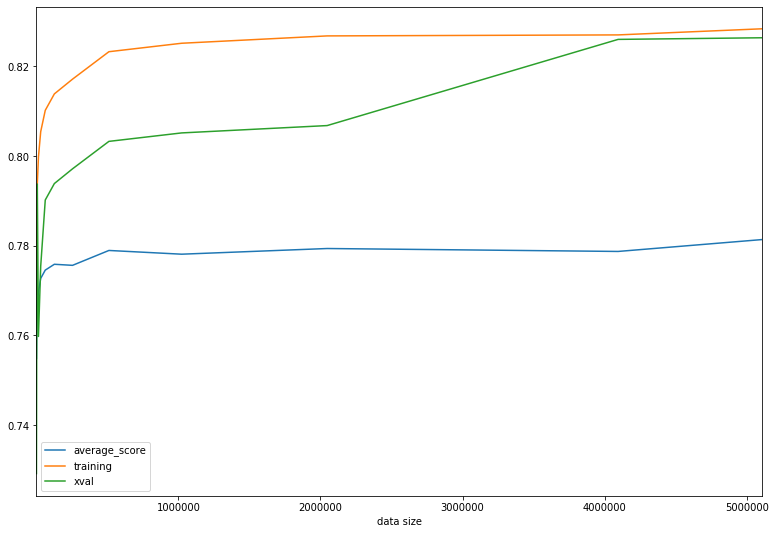
**Eta (learning rate) =** 0.1

**Max\_depth =** 10

* 1. **Curvas de aprendizaje.**

Luego de elegir las mejores configuraciones se procede a probar el algoritmo para diferentes tamaños del dataset. A Continuación se muestran dos curvas, una con el accuracy y otra con el Mean Average Accuracy (M.A.A.).

*Score vs numero de samples.*



Para tener un accuracy aceptable: mayor a 0.82, y un M.A.A, mayor a 0.78, se requieren por lo menos 5 millones de sesiones.

**VI. Discusión y conclusiones.**

Se probaron diferentes algoritmos para solucionar el mismo problema. Uno de los desafíos que tuvimos, fué el de preprocesar los datos, el cual fué dado por por diferentes motivos. El primero fué poder lidiar con grandes cantidades de datos (el Training set superaba los 100GB) y poder manipularlos con los recursos computacionales que poseemos; para ello decidimos reducir la data usando muestras aleatorias de todas las sesiones, con lo cual redujimos los datos drásticamente (alrededor de 4 GB para training y 500 MB para testing), y, nuestra principal preocupación era perder robustez en los modelos. El segundo era el poder lidiar con data secuencial, debido a que si una canción se omite o no, depende mucho de las canciones anteriores.

Todos los algoritmos que se usaron, presentaron un rendimiento aceptable respecto al accuracy (mayor a 0.8), sin embargo Spotify no usa el accuracy para obtener el groundtruth, sino más bien el mean-average-accuracy (MAA) que es calculado por cada sesión, en el cual los ensembles (Bagging y Boosting) son los ganadores; M.A.A alrededor de 0.783 y un accuracy alrededor de 0.852.

Para evaluar un algoritmo no solo basta con los resultados de las predicciones, si no también con la eficiencia de training y testing, en las cuales Boosting (xgboost) fué el vencedor. Bagging podría hasta triplicar el tiempo de traininig y testing, respecto a xgboost.

Con respecto a naive-bayes, la principal ventaja, es que cual sea la configuración con la que se entrene, no mejora ni empeora en rendimiento significativamente. Pero debido a que no se calculó el MAA, no se puede hacer una comparación más a fondo. Y por último, el decision tree también muestra un buen rendimiento, pero es muy susceptible a overfitting si se entrena con muchos datos, o underfitting si no se seleccionan los parámetros adecuados.

Nuestra solución al challenge para lidiar con datos secuenciales condición simplemente en entrenar los modelos con la canción actual y la anterior, pero la desventaja de esto, es que no se toma en cuenta la información de canciones anteriores, por lo que se pierde información valiosa. Una solución para este problema podría ser usar LSTM, el cual es un algoritmo diseñado especialmente para lidiar con datos secuenciales y series de tiempo.