**Course Approve Detection Classifier**

En este archivo se explicará el proceso y la lógica que se empleó para resolver el ejercicio desafío que se me propuso.

Se comenzó con el ejercicio propuesto realizando un pequeño EDA con las distintas variables presentes en el CSV. Se vio la distribución de todas las variables, los posibles valores que estas podían tomar y el tipo de datos que contenían las distintas columnas, sin perder de vista el problema a resolver.

**Consideración:** Algo importante a mencionar antes de comenzar a detallar el proceso de transformación de los datos, es que, en la consigna se pedía realizar un índice compuesto por el **id alumno** / **id materia** / **partición**. Si bien dada la naturaleza de los datos esto parecía ser lo más apropiado, decidí no incluir la partición en el índice. Eso lo hice así ya que no creía correcto considerar una **variable temporal** en el índice, ya que cada partición generaba una repetición de registros de cada alumno y curso.

Dicho de otra manera, me resultaba más complejo evaluar la performance de un único alumno, cuando la información respecto a la performance de este mismo se encontrada dividida en varias variables temporales.

Por lo tanto, lo que decidí hacer es trabajar los datos de manera que cada fila del Data Set fuera un único alumno en un determinado curso, así de esa manera se tendría toda la información del alumno en una única fila y sería más fácil crear un modelo para detectar si ese alumno iba a aprobar o no determinado curso.

Echa esta aclaración comenzare a describir como fue el proceso de transformación de los datos.

**Data Wrangling:**

Lo primero que se hizo fue crear una copia nueva del Data Set original, en donde el índice estaba compuesto por **id alumno / id materia**.

Por lo que una única fila se correspondía con un alumno en determinada materia, un id de alumno podía aparecer varias veces (un alumno inscripto en distintas materias) así lo mismo con las materias (varios alumnos realizando una materia).

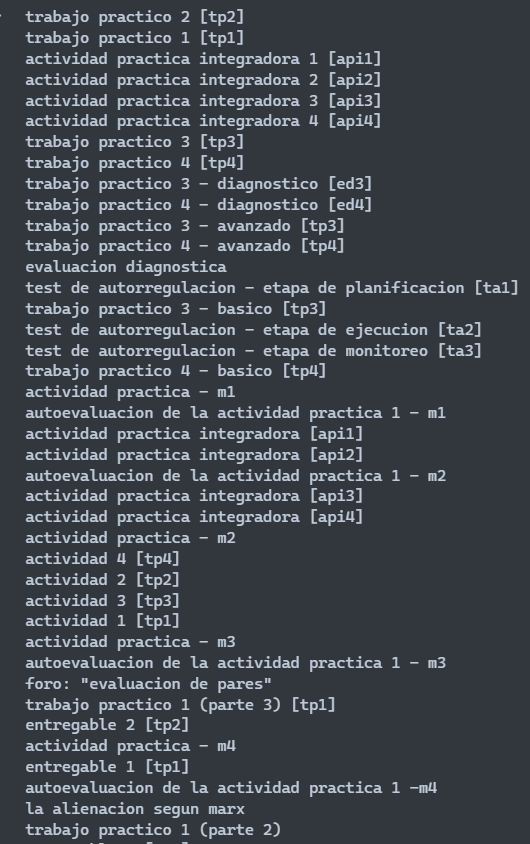
Agrupe en base al periodo de la materia y en base a la nota final de cada alumno. La nota final se corresponde a la **variable target** del modelo de clasificación.

Una vez generado el índice lo que hice fue tratar de trabajar la información disponible de manera que se pueda obtener la mayor información con los datos que se tenía.

**Trabajos Prácticos:**

Primero agregue la información en base a los trabajos prácticos lo que busque acá era ver que trabajos prácticos había entregado cada alumno y que nota había obtenido, además de ver que trabajo practico se correspondía a cada materia, porque no todas las materias tenían la misma cantidad y tipo de trabajos prácticos.

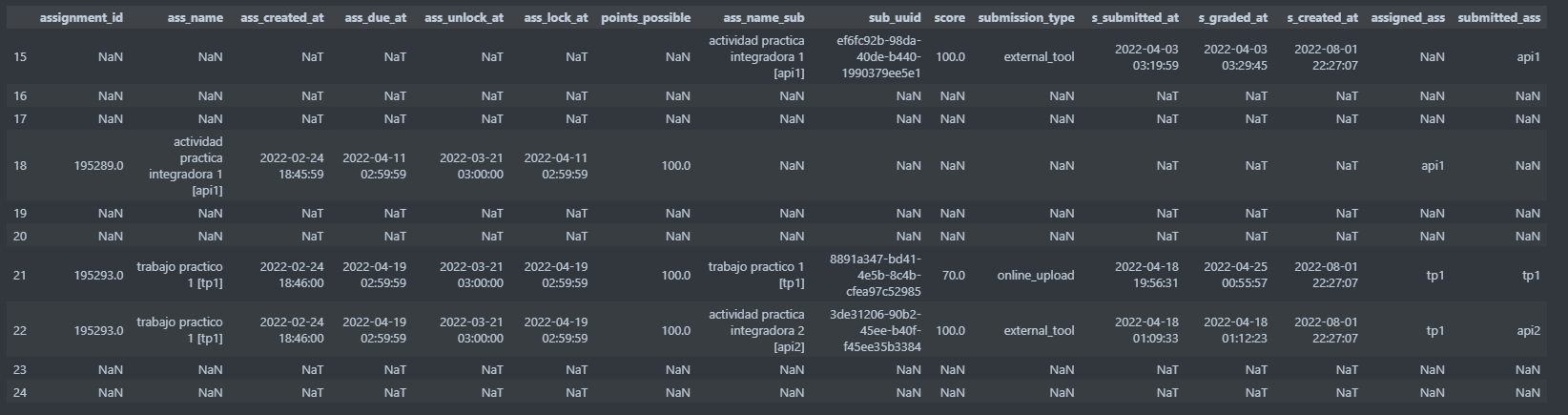
Las variables que se utilizaron para obtener esta información fueron “**ass\_name**” y “**ass\_name\_sub**”, viendo los distintos nombres que tomaban los distintos trabajos vi que podía unificarse los trabajos practicos en una única consideración.



Se crearon dos variables nuevas (“**assigned\_ass**” que se obtuvo de la columna ass\_name y “**submitted\_ass**” se obtuvo de la columna ass\_name\_sub), en las cuales mediante **regex** se extraía de los distintos valores de tipo de trabajo practico, este mismo se encontraba encerrado entre corchetes “[….]”.

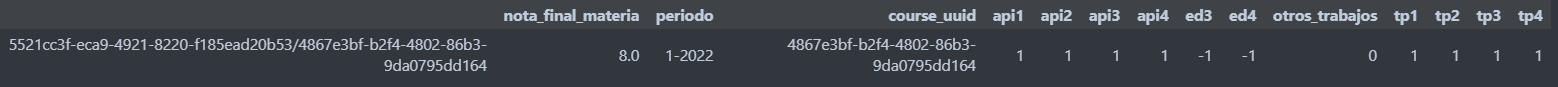
Se agruparon los trabajos prácticos en varias categorías: TP1, TP2, TP3, TP4, API1, API2, API3, API4, ED3, ED4 (no figuraban valores de ED1 y ED2).

Luego se hizo la consideración de que, **si los trabajos prácticos solo tenían nombre, pero no el tipo** (“entregable grupal 1” por ejemplo) se los consideraría dentro de una categoría llamada “**otros\_trabajos**”.

Al crear las dos variables nuevas y mientras analizaba los datos obtenidos me di cuenta que en la **partición** de la asignación de un trabajo practico no necesariamente correspondida con la entrega del mismo, y que en una misma partición podía haber con un mismo nombre de trabajo asignado otro tipo de trabajo entregado. Además de que la entrega de un trabajo podía tener otro nombre diferente al asignado. como en el ejemplo de abajo.

**Consideración:** Debido a que la variable **ass\_name** y **assigned\_ass** repetían valores varias veces y no se correspondían con las tareas que los distintos alumnos entregaban se utilizó la variable **submitted\_ass** para obtener los trabajos prácticos entregados por los alumnos, y se verifico con los trabajos asignados que, los trabajos entregados se correspondan a los que se asignaban en cada materia.

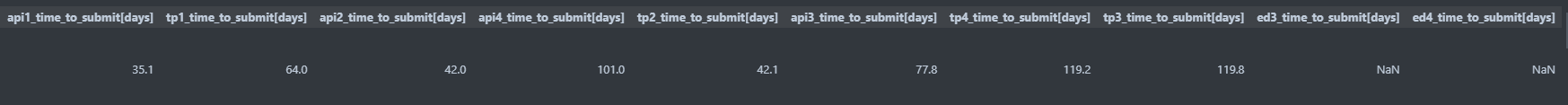
Los trabajos prácticos que no eran asignados en una materia se los flaggeo con un valor de -1. Por lo tanto, los posibles valores que podía tener cada entrega de trabajo practico era:

* 1 a n (n siendo un numero entero) ­– Si el trabajo practico se asignó en la materia y el alumno lo entrego. Si un trabajo practico era entregado varias veces n se correspondía con la cantidad de veces.
* 0 – Si el trabajo practico era asignado, pero no entregado.
* -1 – Si el trabajo practico no se correspondía con la materia considerada.

Aplicando la misma lógica que para lo anterior obtuve también el **puntaje** de cada alumno en la entrega de cada trabajo.

Luego de obtener la información mencionada, trabaje con las variables temporales para obtener el **tiempo que demora cada alumno en entregar cada trabajo.**

Para esto primero obtuve la fecha en la que se asignaba cada trabajo practico (columna ‘**ass\_created\_at**’) en cada curso. Luego obtuve la fecha en el cual cada alumno entrego (columna ‘**s\_submitted\_at**’) los trabajos prácticos de cada materia.

 Con estas dos fechas fui capaz de obtener el tiempo que transcurrió desde la asignación, hasta la entrega de cada trabajo practico.

**Formato de entrega:**

El siguiente paso del Data Wrangling fue obtener la cantidad de entregas de cada formato que un alumno había realizado, siguiendo la misma lógica (agrupando por el índice y obteniendo las cantidades de formatos entregados de cada alumno).

Los posibles tipos de entrega son: 'external\_tool', 'online\_upload', 'online\_quiz', ‘discussion\_topic', 'basic\_lti\_launch', 'online\_text\_entry', 'media\_recording'.

**Exámenes:**

La última información que se agregó a la data final fue la relevante a los exámenes.

**Consideración:** En la columna ‘ass\_name\_sub’ (se corresponde a los trabajos prácticos entregados) contenía un valor que hacía referencia a una ‘evaluación diagnostica’, me pareció que esta evaluación era importante a la hora de determinar si un alumno aprobaba o no una materia. Debido a esto, agregue a la misma como una feature para el modelo.

Luego de agregar las notas, lo que hice fue flagear con -1 aquellos exámenes que el alumno no hubiese rendido, ya que si los flageaba con 0 podría estar imputando un valor erróneamente (como si el alumno hubiera obtenido un 0 en dicha evaluación).

**Limpieza registros sin exámenes.**

Por último, una vez que ya generé las variables de interés necesarias, procedí a hacer una limpieza de datos. Pude observa que había alumnos que no tenían ningún examen rendido, esto podía deberse a algún tipo de error en la carga de datos o algún error externo que no es de mi conocimiento. Ya que podían afectar negativamente la performance del modelo, decidí eliminar estos registros.

**Exploratory Data Analysis:**

Una vez obtenidas las variables que se iban a utilizar para entrenar al modelo se procedió a realizar un Análisis Exploratorio de Datos, para conocer el comportamiento de las distintas variables creadas.

Busque analizar estas variables contra la variable target, viendo correlaciones y distribuciones. Se observo que algunas variables tenían sentido como se comportaban frente a la variable de interés.

En esta etapa no buscaba conocer grandes relaciones dentro de los datos, sino que validar de alguna manera que las variables creadas tuvieran sentido y se comporten como lo esperaba.

**Modelo de clasificación:**

Luego de tener un mejor conocimiento de los datos se procedió a crear una notebook en donde realice distintas pruebas y entrenamientos de modelos de clasificación.

Lo primero que se hizo fue preparar los datos:

* **Separar datos en train y test –** Se separaron los datos en train y test, con los datos de train se entrenaron los distintos modelos y con los datos de test se valido la performance de los mismos.
* **Normalizar –** Se llevaron las diferentes variables numéricas creadas a una misma escala.
* **Balanceo de clases –** Se pudo observar que la clase target estaba altamente desbalanceada, por tal razón una alternativa para mejorar la performance del modelo fue aplicar un balanceo de clases.  
  Se opto por utilizar **Oversample** ya que la clase negativa presentaba pocos casos, entonces si se aplicaba Undersample el dataset resultado iba a tener dimensiones muy chicas, y el modelo probablemente no iba a poder “aprender” de los mismos.

Una vez que los datos estaban listos para ser consumidos por un modelo de Machine Learning se propusieron tres modelos para comparar:

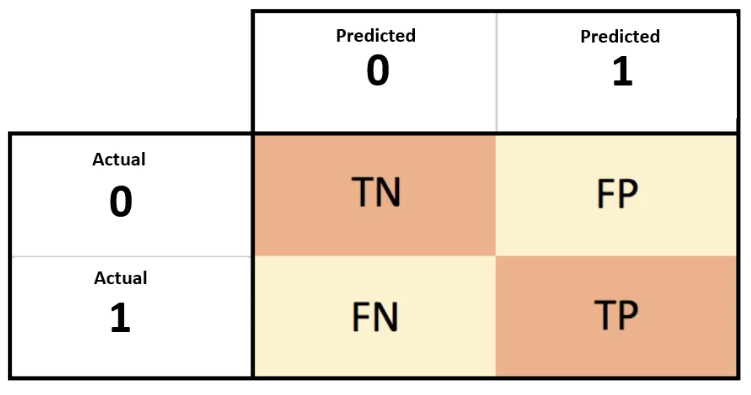
* Regresion Logistica.
* Random Forest.
* XGBoost (Gradient Boosting).

**Consideración:** Cabe aclarar que además de compararse estos modelos entre sí, compare la performance de los mismos al ser entrenado en dos escenarios, con los datos desbalanceados y con los datos balanceados.

**Técnicas de evaluación de modelos:**

Se entrenaron y evaluaron los modelos mencionados utilizando las siguientes métricas:

* **Accuracy –** Evalúa la precisión global del modelo**.**
* **Recall –** Capacidad del modelo de detectar correctamente los casos positivos, proporción de casos positivos que se identifican correctamente con respecto a todos los positivos reales.
* **Precision –** Precisión de las predicciones positivas, indica la proporción de casos positivos identificados correctamente por el modelo en relación a todos los positivos predichos (True positive y False positive)**.**
* **Specificity –** Capacidad del modelo para identificar correctamente los ejemplos negativos**.**
* **F1-Score –** Media armónica entre precisión y recall, es útil cuando hay un desequilibrio en la distribución de clases.
* **Curva ROC –** Representación grafica del rendimiento de un modelo. Muestra la tasa de verdaderos positivos (**TPR**) en función de la tasa de falsos positivos (**FPR**) para diferentes valores de umbral. Mientras esta curva este más cercano al área superior izquierda del gráfico, mejor será el rendimiento del modelo.
* **AUC –** Representa el área bajo la curva ROC. Esta métrica evalúa la capacidad de un modelo para distinguir entre clases positivas y negativas.
* **Confusion Matrix –** Es una matriz que resume el rendimiento del modelo, mostrando la cantidad de ejemplos clasificados correctamente e incorrectamente para cada clase.



**Evaluación de modelos:**

Al ver las métricas y observar la curva ROC, se determinó que de los 6 modelos planteados el que mejor preformo fue **XGBoost**.

Algo que mencionar es que, si bien el modelo mencionado fue el de mejor performance, se observó que todos los modelos tenían complejidades al detectar la clase negativa (alumnos que desaprobaban los cursos), pero no así la clase positiva (si el alumno aprobaba el curso).

Por lo tanto, busque una solución para poder mejorar la detección de los verdaderos negativos, esto lo logre mediante la modificación del umbral de decisión.

Este umbral mide la probabilidad de pertenencia de cada observación a cada clase, para determinar que clase se asigna dicha observación.

Lo que hice fue aumentar el umbral de manera de priorizar una alta sensibilidad para tratar de atrapar la mayor cantidad de verdaderos negativos que se pueda. Al hacer esto estamos aumentando también la cantidad de falsos positivos que el modelo va a detectar, pero dada la naturaleza del problema y lo que se quiere identificar es algo que estoy dispuesto a sacrificar.

**Consideración:** Basado en lo que dije anteriormente, voy a establecer que para la solución de este problema es mas **importante detectar si un alumno desaprueba el curso**, por lo tanto, todas las técnicas que emplearemos para mejorar el modelo tendrán como finalidad, la mencionada.

**Modificación del umbral de decisión:**

Para obtener el nuevo umbral de decisiones lo que hice fue mirar la curva ROC, centrándome en el modelo que seleccione como mas opimo, me fije el punto que se encontraba más próximo a la esquina de arriba a la izquierda.

Del punto mencionado obtuve el True Positive Rate y el False Positive Rate. Con esos valores obtuve el umbral de decisiones según la curva ROC del modelo.

Al observar la performance del modelo pudimos ver que efectivamente se mejor la predicción de la clase negativa, pero apareció en mayor presencia la detección de falsos negativos, como era de esperarse.

Este modelo fue el que mejor performo de todos, ya que tenia bastante exactitud para predecir si un alumno aprobaba o desaprobaba el curso.

**Randomized Grid Search:**

Además de modificar el umbral de decisión, trate de buscar la combinación más optima de hiperparámetros.

Para esto utilice **Randomized Search CV**, es una técnica que ayuda a encontrar los mejores hiperparámetros para un modelo de ML.

Elegí este método ya que **Grid Search CV** es más costoso computacionalmente, debido a que realiza una búsqueda exhaustiva de todas las combinaciones posibles de hiperparámetros que se le pase al modelo.

En Randomized, en lugar de evaluar todas las combinaciones posibles, se seleccionan aleatoriamente algunas configuraciones, lo que es útil cuando se desea ahorrar tiempo de cómputo.

**Consideración:** Si bien el Dataset de entrenamiento era chico y no implicaba un alto costo computaciones, decidí usar randomized debido a que al probar con GridSearch se demoraba mucho el tiempo de ejecución.

Una vez corrido RandomizedSearchCV obtuve los mejores hiperparámetros y compare el modelo contra el anterior (el que se le modifico el umbral de decisión). Este modelo no obtuvo una mejora significativa en la detección de verdaderos positivos.

De igual manera que antes modifique el umbral de decisión para ver si podía mejorar la detección de verdaderos positivos, pero para este caso al hacer esto, aumento en gran medida la cantidad de falsos negativos detectados, en comparación al modelo anterior.

Por lo tanto, se procedió a seleccionar como el modelo de mejor performance al modelo que, se **balanceo la clase target**, se entrenó con los **hiperparámetros baseline** y se le **modifico el umbral de decisión**.

Para finalizar se exporto el modelo entrenado utilizando la librería joblib, dejando el modelo listo para su utilización y predicción. La consideración que hay que tener es que la modificación del umbral no puede ser configurada en el entrenamiento del modelo. Por esta razón cuando se realicen las predicciones con el modelo entrenado, deberá previamente corregir el umbral de decisión para que las predicciones se ajusten a lo mencionado.

**Dockerizacion:**

Luego de finalizada la selección del modelo se procedió a realizar un empaquetado con las distintas dependencias, con el objetivo de crear simplificar la ejecución del modelo.

El Docker generado se encarga de tomar como input los datos del challenge, luego se les aplica un **data wrangling** para transformar los datos y obtener así los inputs del modelo.

Un segundo archivo se ejecuta luego de que el data wrangling haya concluido, este se encarga de tomar los datos y entrenar el modelo de clasificación ganador, además de aplicarle la modificación del umbral de decisión.

El output de esta segunda notebook consta de 2 archivos, un CSV con las predicciones del modelo (los datos de test), y luego un archivo ‘.joblib’ que contiene el modelo entrenado listo para predecir.

Para ejecutar el Docker file es necesario, utilizando la terminal, ir hasta la carpeta de destino donde se encuentre el archivo con la extensión ‘.tar’, una vez ubicado en destino, se ejecuta el siguiente comando:

**docker load -i course\_approve\_detection\_dock.tar**

Luego de que se carga la imagen, se ejecuta la siguiente línea de código:

**docker run -v /path\_to\_host/output:/app/output course\_approve\_detection\_dock**

**Consideración:** en donde dice ‘path\_to\_host’ hay que especificar el directorio en donde se esta corriendo la imagen, así de esa manera no hay errores con los outputs del modelo.

**Integración:**

Para integrar este modelo en un producto útil para los usuarios de Ed Machina, hay que considerar varios aspectos:

**Integración en plataforma,** se podría desarrollar una API que permita la comunicación entre la plataforma de Ed Machina y el modelo de clasificación.

**Interfaz de Usuario (UI),** diseñar una interfaz de usuario que permita a los mismos interactuar fácilmente con el modelo.  
 En ella se debería proporcionar información clara sobre las predicciones del modelo, como la probabilidad de aprobación y factores que influyen en la predicción.  
 Además de incorporar visualizaciones interactivas para que los usuarios puedan explorar y comprender mejor los resultados.

**Experiencia del Usuario (UX),** asegurarse de que la integración del modelo no afecte negativamente la experiencia del usuario.  
 Podría proporcionarse retroalimentación útil sobre por que se hizo determinada predicción, para ayudar a los usuarios a comprender las decisiones del modelo.

**Notificaciones y Alertas,** implementación de un sistema de notificaciones para informar sobre las predicciones del modelo, podría notificarse cuando se predice que un alumno podría reprobar el curso, así de esa manera el profesor puede tomar medidas para revertir esa situación.  
 Podrían implementarse notificaciones personalizadas, para que sean informativas, brindando sugerencias para mejorar el rendimiento académico.

**Iteración Continua,** monitorear continuamente el rendimiento del modelo y realizar actualizaciones según sea necesario, para mantener la eficacia a lo largo del tiempo.Recopilar experiencias de los usuarios al utilizar la interfaz o la precisión de las predicciones para mejorar la performance y utilidad del modelo.