**PROYECTO DATA SCIENCE BASADO EN EL MUNDO DEL TENIS**

**OBJETIVO**

El objetivo principal de este proyecto es explorar una serie datos obtenidos referentes a un histórico de partidos de tenis que va desde 1960 hasta principios de 2018, transformar dichos datos para aplicar una serie de modelos de machine learning que nos puedan dar evidencias claras de cómo se comporta el mundo del tenis y así poder predecir nosotros mismos quién va a ser el ganador de un partido de Grand Slam.

**DESCRIPCCIÓN DE LOS DATOS**

He utilizado un dataset obtenidos de Kaggle, que provienen con total seguridad de un scrapeo realizado por otra persona en la página oficial de la ATP World Tour, web que me ha servido para contrastar que la información que he ido manejando era correcta:

Fichero de raw data utilizado:

* all\_matches.csv: dataset de 54 columnas y 787MB, este es el dataset principal y contiene información de cada partido como aces, ganador, puntos de break en contra, etc.

**HELLO DATA SET**

En primer lugar, hacemos un head para explorar los datos y ver cómo están distribuidos, ver que encoding tienen, etc.

Filtramos el Dataset por “end date”, “location” y “player id” y examinamos un poco todos los datos que nos ofrecen las diferentes columnas del dataset y nos damos cuenta de que cada partido aparece dos veces ya que cada uno hace referencia a los datos de uno de los dos jugadores, es decir, la final de Wimbledon de 2008 tendrá dos filas, una que habla de las estadísticas de Federer y otra que hablará de las estadísticas de Nadal, por lo que ya tenemos una idea de cómo empezar el limpiado de los datos.

Fichero utilizado:

* Hello\_Dataset.ipynb

**DATA CLEANING**

Utilizamos pandas para el procesado, limpieza, filtros y transformación de los datos.

A su vez, establecemos las reglas de negocio, por lo que decidimos filtrar solo por partidos de Grand Slam, a partir de 1990, partidos individuales y nos quitamos variables que no sirven como “prize money”, “masters”, “nation”, “currency”, “location”…

Vemos que había partidos que no tenían datos, nos los quitamos y con el “.copy “ nos aseguramos de que es una copia nueva en memoria.

Le asignamos número 1000 a la variable “seed” porque al darle este número muy elevado el modelo que vayamos a utilizar sabrá diferenciar mejor, rellenamos los nulos con 1000 y luego pasamos a entero toda la columna.

Tratamos los players para tener la información de los jugadores en la misma fila: seleccionamos el “player\_id” y el “opponent\_id”, los juntamos con un espacio los nombres ordenados del “player” y del “opponent”. En las siguientes líneas usamos “head(1)” para coger el partido del gropby por player y por “tail(1)” para coger el partido del groupby por opponent para posteriormente hacerle el “merge” por “player”, “torneo” y “año”.

Finalmente quitamos los totales y demás información irrelevante, porque eso solo lo queremos para el “df\_matches\_grand\_slam\_cleaned” pero no para el “df\_players\_cleaned”, porque el total tiene en cuenta también los puntos del otro.

Ficheros utilizados:

* Data\_cleaning.ipynb
* Matches\_grand\_slam\_cleaned.csv
* Players\_cleaned.csv

**EXPLORATORY DATA**

En esta parte utilizamos “seaborn” y “matplotlib” para realizar la visualización de los datos,.

Visualizamos varias de las correlaciones que se establecen entre los datos, y vemos que puede existir una correlación importante entre los saques directos (aces) que hace un jugador y los partidos ganados, en dicha visualización se ve un outlier, por lo que estudiamos el caso y vemos que efectivamente es un partido de Wimbledon en 2010 entre Isner y Mahut, que curiosamente resulta ser el partido más largo de la historia, por lo que queda totalmente entendida su condición de outlier.

Posteriormente comprobamos si la relación entre aces y juegos ganados es diferente para los distintos torneos, asignamos un color a cada torneo y visualizamos dicha correlación, vemos como la pendiente de Roland Garros es mayor, y eso es debido a que hacen muchos menos aces, pero cuando se hacen muchos aces en tierra batida la probabilidad de ganar más juegos es mayor.

Ficheros utilizados:

* matches\_grand\_slam\_cleaned.csv
* Exploratory\_Data\_Analysis.ipynb

**CLUSTERING JERÁRQUICO**

He decidido aplicar este modelo para ver diferencias entre los jugadores.

Para aplicar un clustering jerárquico a nuestros datos, debemos:

* Importar dendrogram, linkage de scipy.cluster.hierarchy
* Transformar las columnas a numérico.
* Agrupar los datos totales históricos de cada jugador.
* Seleccionamos los 25 mejores jugadores diferenciando como mejores, los que más victorias tienen.

Finalmente tenemos un Dendrograma (diagrama de árbol que muestra los grupos que se forman al crear conglomerados de observaciones en cada paso y sus niveles de similitud. El nivel de similitud se mide en el eje vertical y las diferentes observaciones se especifican en el eje horizontal).

Es interesante ver como el modelo a establecido a Federer como jugador más diferenciado del resto, seguido directamente por Djokovic y Nadal.

Ficheros utilizados:

* players\_cleaned.csv
* matches\_grand\_slam\_cleaned.csv
* Clustering.ipynb

**CLUSTERING KMEANS**

Aplicamos este modelo para ver similitudes y diferencias entre los jugadores.

Para aplicar un clustering Kmeans debemos:

* Importar KElbowVisualizer de yellowbrik.cluster
* Importar Kmeans de sklearn.cluster

Utilizamos el “método del codo” para elegir el número óptimo de clusters, vemos que es 5. Una vez realizado el clustering vemos como el modelo ha sido capaz de diferenciar ocho grupos de jugadores basándose en la media de victorias.

El clúster número 4 contiene solo a ocho jugadores y como podemos ver tras hacer un display de lo que hay en cada clúster, esos son prácticamente los mejores jugadores de la historia, el modelo ha sabido diferenciar quienes son los mejores y quienes son los no tan buenos, como por ejemplo, el clúster número 7 contiene a jugadores tipo Tommy Robredo, cuya trayectoria ha sido notable pero no tan estratosférica como la de los jugadores del clúster 4.

Ficheros utilizados:

* players\_cleaned.csv
* matches\_grand\_slam\_cleaned.csv
* Clustering.ipynb

**MODELO SUPERVISADO XGBOOST**

Aplicamos este modelo porque es uno de los mejores modelos predictivos con datos tabulares.

Para aplicar el modelo XGBoost debemos:

* Importar XGBClassifier de xgboost
* Importar plot\_importance de xgboost
* Importar pyplot de matplotlib
* Importar train\_test\_split de sklearn.model\_selection
* Importar roc\_auc\_score, accuracy\_score de sklearn.metrics
* Pasar el target a numérico ya que este modelo sólo puede tener datos en numérico.
* Generar un dataframe que tenga las estadísticas acumuladas de los dos jugadores, lo llamamos df\_matches\_subset\_join.
* Quitamos las variables categóricas tipo nombre, fecha, año y ronda.
* Dividimos los datos de train y los datos de test para poder testear que el modelo funciona correctamente.
* Params\_xgb:
  + 200 árboles
  + Profundidad 4
  + Reg\_lambda = 15, esto hace que el árbol tenga menos nodos porque busca que cada Split del árbol tenga una ganancia determinada, sirve para evitar overfiting.
  + Mind\_child\_weight = 10, número mínimo de tenistas que caigan en un mismo nodo, para reducir overfiting.

Ejecutamos el modelo y vemos como funciona bien, tiene una precisión cercana al 70% y además es un poco más baja en train que en test.

Finalmente podemos ver como al pasarle al modelo la diferencia de victorias, tiene un poco mejor rendimiento que al pasarle la variable dividida en player\_victory\_1 y player\_victory\_2.

Métricas utilizadas para evaluar el modelo:

* AUC: mide cómo de correctamente el modelo nos ordena el score.
* Accuracy: qué porcentaje de aciertos tiene el modelo.

Ficheros utilizados:

* Modelo\_Supervisado.ipynb
* players\_cleaned.csv
* matches\_grand\_slam\_cleaned.csv

**INTERFACE**

El usuario debe introducir los nombres de los jugadores cuyo enfrentamiento quiere predecir (Primera letra del nombre seguido de un puto y el apellido, ejemplo: R. Nadal).

Nos devolverá quién ganará, con qué probabilidad y qué tres variables son las que más han influido en ese resultado.

Ficheros utilizados:

* data.pkl
* model.pkl
* shap.explainer.pkl