prog_datasci_7_analisis_entrega

May 30, 2023

1 Fundamentos de Programación

1.1 PEC 7 - Introducción al análisis de datos en Python

En este Notebook se encontraréis el conjunto de actividades evaluables como PEC de la asignatura. Veréis que cada una de ellas tiene asociada una puntuación, que indica el peso que tiene la actividad sobre la nota final de la PEC. Adicionalmente, hay algún ejercicio opcional, que no tiene puntuación dentro de la PEC, pero que se valora al final del semestre de cara a conceder las matrículas de honor y redondear las notas finales. Podréis sacar la máxima nota de la PEC sin necesidad de hacer estos ejercicios. El objetivo de estos ejercicios es que sirvan como pequeño reto para los estudiantes que quieran profundizar en el contenido de la asignatura.

Veréis que todas las actividades de la PEC tienen una etiqueta, que indica los recursos necesarios para llevarla a cabo. Hay tres posibles etiquetas:

- NM **Sólo materiales**: las herramientas necesarias para realizar la actividad se pueden encontrar en los materiales de la asignatura.
- EG **Consulta externa guiada**: la actividad puede requerir hacer uso de herramientas que no se encuentran en los materiales de la asignatura, pero el enunciado contiene indicaciones de dónde o cómo encontrar la información adicional necesaria para resolver la actividad.
- EI **Consulta externa independiente**: la actividad puede requerir hacer uso de herramientas que no se encuentran en los materiales de la asignatura, y el enunciado puede no incluir la descripción de dónde o cómo encontrar esta información adicional. Será necesario que el estudiante busque esta información utilizando los recursos que se han explicado en la asignatura.

Es importante notar que estas etiquetas no indican el nivel de dificultad del ejercicio, sino únicamente la necesidad de consulta de documentación externa para su resolución. Además, recordad que las **etiquetas son informativas**, pero podréis consultar referencias externas en cualquier momento (aunque no se indique explícitamente) o puede ser que podáis hacer una actividad sin consultar ningún tipo de documentación. Por ejemplo, para resolver una actividad que sólo requiera los materiales de la asignatura, puedéis consultar referencias externas si queréis, ya sea tanto para ayudaros en la resolución como para ampliar el conocimiento!

En cuanto a la consulta de documentación externa en la resolución de los ejercicios, recordad **citar siempre la bibliografía utilizada** para resolver cada actividad.

1.1.1 Ejercicio 1

En la primera parte de la PEC vamos a trabajar con dos datasets, `movie.csv` y `ratings.csv`, que se encuentran en la carpeta `data`. **(2 puntos)**

a) Carga los dos datasets e imprime por pantalla sus dimensiones, así como el nombre de las columnas. Muestra también las 5 primeras filas de cada dataset. NM **(0.25 puntos)**

[ ]:
```
# Respuesta
```

b1) En el dataset movie, hay información sobre el año de la película en la columna `title`. Extrae la fecha de la película y crea una nueva columna con esta información.

¿En cuantas películas no tenemos información sobre el año? Muestra las filas por pantalla.

Nota: Ten en cuenta que solo nos interesa extraer los números entre paréntesis. Si hay algún texto, no lo queremos. Revisa también que los valores tengan sentido, si hay años que no sean posibles, sustitúyelos por NaNs.

EG **(0.5 puntos)**

[ ]:
```
# Respuesta
```

b2) Por otro lado, en la columna de `genres`, hay películas con más de un género. Para simplificar, queremos quedarnos solo con el primero. Crea una nueva columna con esta información.

NM **(0.25 puntos)**

[ ]:
```
# Respuesta
```

b3) Muestra las 5 primeras filas del dataset movie con las 2 nuevas columnas.

NM **(0.25 puntos)**

[ ]:
```
# Respuesta
```

c1) En el dataset `ratings` tenemos la puntuación que cada usuario ha dado a las películas. Para poder integrar esta información en el dataframe `movies`, calcula la puntuación media por película y añadela como nueva columna al dataframe `movies`.

Nota: Redondea el valor para que solo tenga **2 decimales**.

NM **(0.25 puntos)**

[ ]:
```
# Respuesta
```

c2) Hay alguna película sin puntuación? Cuenta cuantas hay y asígnales un valor de 0. NM **(0.25 puntos)**

[ ]:
```
# Respuesta
```

d) En el dataset de `rating` también podemos extraer cuantas valoraciones tiene cada película. Añade esta información al dataframe de `movies`. NM **(0.25 puntos)**

[ ]:
```
# Respuesta
```

1.1.2 Ejercicio 2

Una vez tenemos el dataset de movie preparado y con la información addicional, vamos a explorarlo un poco. **(1 punto)**

a) Calcula la puntuación media por año y ordénalos de mayor a menor. Muestra el top10 de años (con mayor puntuación) usando un barplot.

NM **(0.5 puntos)**

[ ]:
```
# Respuesta
```

b) Calcula también la puntuación media por género. ¿Qué género tiene la mayor puntuación media, y cuál la peor?

NM **(0.25 puntos)**

[ ]:
```
# Respuesta
```

c) ¿Qué película tiene más valoraciones? Muestra el título de la película y el número de valoraciones.

NM **(0.25 puntos)**

[ ]:
```
# Respuesta
```

1.1.3 Ejercicio 3

En el dataset ampliado de movies tenemos información sobre el año, la categoría, la puntuación y el número de valoraciones por película. Con toda esta información, queremos utilizar un algoritmo de clustering para agrupar las películas en diferentes grupos según sus características. Para lograrlo, vamos a utilizar el algoritmo **KMeans**.

(**2 puntos**)

a) Primero de todo, vamos a quedarnos solo con las características que nos interesan, en este caso, `movieId`, `year`, `principal_genre`, `rating` y `userId`. Muestra el nuevo dataframe por pantalla. NM (**0.25 puntos**)

[ ]:
```
# Respuesta
```

b) Para usar el algoritmo de KMeans, no podemos tener NaNs. Comprueba si hay NaNs en el dataset. ¿Qué estratégias puedes llevar a cabo para solucionar este problema? Da 2 posibilidades, razona cuál escogerías en este caso y aplícala. NM **(0.25 puntos)**

[ ]:
```
# Respuesta
```

c) Los elementos de la columna del género principal, al ser un string, no se pueden usar directamente en el KMeans, sino que se tienen que transformar a valor numérico. ¿Qué tipo de transformación es más adecuada para esto? Aplícala. NM **(0.25 puntos)**

```
[ ]: # Respuesta
```

d) Antes de aplicar el KMeans, normaliza los datos numéricos utilizando la función MinMaxScaler. ¿Por que razón es importante hacer esta transformación? NM **(0.25 puntos)**

```
[ ]: # Respuesta
```

e) Uno de los problemas de los métodos no supervisados es identificar el número óptimo de clusters. Para poder estimar esta número óptimo, se utiliza frecuentemente el método de Elbow. Busca información sobre este método y utilízalo. ¿Cuál es el número óptimo de clusters? Considera un rango entre 1 y 20 clusters. EI **(0.5 puntos)**

```
[ ]: # Respuesta
```

f) Aplica el método KMeans especificando el número óptimo de clusters obtenido en el apartado anterior. Cuantas películas hay en cada cluster? NM **(0.5 puntos)**

```
[ ]: # Respuesta
```

1.1.4 Ejercicio opcional

Una plataforma de contenido digital nos ha contactado para que le ayudemos a hacer un algoritmo que permita recomendar películas, basándonos en las valoraciones de los usuarios de esta plataforma. Para demostrar que estamos preparados para tal desafío, vamos a usar los datos de películas y valoraciones del ejercicio anterior para construir un sistema de recomendación.

a) Utilizando el dataset de `ratings.csv` original, crea una `pivot table` que tenga **movieID** como filas, **userID** como columnas, y las valoraciones (**rating**) como el valor. Esta nueva tabla nos dará información de la valoración que ha puesto cada usuario a cada película.

```
[ ]: # Respuesta
```

b) Los usuarios no puntúan todas las películas, por eso tenemos bastantes NaNs. Sustituye estos NaNs por 0.

```
[ ]: # Respuesta
```

c) Como podéis ver, la matriz resultante tiene muchos 0, ya que hay muchas películas que no se puntúan. Para que esto no nos afecte en el análisis, vamos a quedarnos solo con aquellos usuarios que han puntuado más de 10 veces y aquellas películas valoradas más de 50 veces.

```
[ ]: # Respuesta
```

d) Para construír el algoritmo de recomendación, necesitamos comprimir el dataframe en una matriz esparza (*sparce matrix*). Para hacerlo, vamos a usar la función csr_matrix.

[ ]:
```
# Respuesta
```

e) A continuación, haz un reseteo de los índices (con **reset_index**) del dataframe anterior, ya que vamos a necesitarlo en los siguientes apartados.

[ ]:
```
# Respuesta
```

f) Finalmente, vamos a entrenar el algoritmo no supervisado de K-Nearest Neighbors. Como parámetros, vamos a usar la metrica `cosine` y 20 vecinos (*n_neighbors*).

[ ]:
```
# Respuesta
```

g) Para comprobar que hemos seguido los pasos correctos, vamos a usar la función `get_movie_recommendation` para sacar 5 películas recomendadas basadas en **Matrix**. Estudia bien la función para entender cuáles son los inputs necesarios y como se relacionan con los pasos que hemos hecho anteriormente.

[ ]:
```
def get_movie_recommendation(
    movies_df,
    user_movie_ratings_df,
    movie_name,
    n_movies_to_reccomend,
    model
):
    """
    movies_df: DataFrame de películas original (es decir, movie.csv)
    user_movie_ratings_df: DataFrame de la pivot table después del reseteo de los índices
    movie_name: nombre de la película de la cual queremos recomendaciones
    n_movies_to_reccomend: número de recomendaciones (e.g. 10)
    model: modelo NN entrenado con la matriz esparza (csr_matrix)
    """
    # Buscamos películas que contengan el nombre que hemos indicado
    movie_list = movies_df[movies_df['title'].str.contains(movie_name)]
    # Si hay
    if len(movie_list):
        # Nos quedamos con el índice la 1a película de la lista
        movie_idx= movie_list.iloc[0]['movieId']
        # Mostramos por pantalla la película que vamos a usar como referencia
        print(f"Recommendations for {movie_list.iloc[0]['title']}\n")
        # Miramos a que índice del df de la pivot table corresponde este movieId
        movie_idx = user_movie_ratings_df[user_movie_ratings_df['movieId'] == movie_idx].index[0]
        # Calculamos la distancia y los índices aplicando el KNN a esta película
        distances , indices = model.kneighbors(csr_data[movie_idx],n_neighbors=n_movies_to_reccomend+1)
        # Ordenamos los índices por la distancia para tener los vecinos más similares
```

```
        rec_movie_indices = sorted(list(zip(indices.squeeze().
↪tolist(),distances.squeeze().tolist())),key=lambda x: x[1])[:0:-1]
        # Creamos un lista para guardar los resultados
        recommend_frame = []
        # Vamos a guardar la información de cada película recomendada
        for val in rec_movie_indices:
            # Sacamos el movieId de la película recomendada
            movie_idx = user_movie_ratings_df.iloc[val[0]]['movieId']
            # Miramos a que índice corresponde de movies_df
            idx = movies_df[movies_df['movieId'] == movie_idx].index
            # Guardamos el título y la distancia de la película
            recommend_frame.append({'Title':movies_df.iloc[idx]['title'].
↪values[0],'Distance':val[1]})
        # Transformamos la lista a df
        df = pd.DataFrame(recommend_frame,index=range(1,n_movies_to_reccomend+1))
        return df
    else:
        return "No movies found. Please check your input"
```

[ ]:
```
# Respuesta
```

1.1.5 Ejercicio 4

A continuación vamos a cambiar a un dataset mucho más grande con más películas de todas las épocas. Su nombre es `movies_long.csv` y lo podréis encontrar en la carpeta `data`. **(1.25 puntos)**

(a) Carga el *dataset* e imprime por pantalla sus dimensiones, así como el nombre de las columnas, su tipo y el número de valores no perdidos que contienen. Imprime también las diez primeras filas para ver qué tipo de datos contiene. Elimina las filas que contengan valores perdidos, o NaN, del dataset, para evitar problemas. NM **(0.5 puntos)**

[ ]:
```
# Respuesta
```

(b) Queremos centrarnos en en valor de las ganancias totales de las películas (dadas por la columna 'gross') y ver si podemos encontrar una manera de predecirlo con los datos de los que disponemos. Primero, visualiza la correlación entre todas las variables y grafícala utilizando la librería `seaborn`. Verás que no todas las variables se han añadido en la gráfica. ¿Por qué es? **(0.5 puntos)** NM

[ ]:
```
# Respuesta
```

(c) Muestra una lista con las variables que más se correlacionan con 'gross', ordenada de mayor correlación a menor correlación. NM **(0.25 puntos)**

[ ]:
```
# Respuesta
```

1.1.6 Ejercicio 5

A continuación vamos a intentar utilizar la técnica de PCA para reducir la dimensionalidad del dataset. Hemos visto en el apartado anterior que muchas variables están muy correlacionadas con 'gross', por lo que es bastante probable que exista información redundante. **Nota:** aunque PCA se puede aplicar sin problemas en este caso, realmente no es extremadamente útil, ya que tenemos un número bastante pequeño de "features". PCA es sobre todo útil cuando tenemos muchas dimensiones (o "features") y queremos reducirlo a un número más adecuado para el análisis rápido. En nuestro caso, utilizamos un dataset más pequeño para que sea más sencillo. **(2 puntos)**

(a) Antes de ajustar el modelo de PCA, tipifica los datos con `sklearn.preprocessing.StandardScaler`, para lo que primero tendrás que seleccionar **sólo las columnas que contienen valores numéricos**. ¿Qué hace esta función? Calcula el valor medio y la desviación típica de los datos tipificados **antes y después** de aplicar la función para cada de las variables. **(0.5 puntos)** NM

```
[ ]: # Respuesta
```

(b) Ajusta el modelo de componentes principales sobre los datos tipificados, pero limita el número de componentes principales a 2. ¿Cuánta variabilidad explican estas dos componentes? **(0.5 puntos)** NM

```
[ ]: # Respuesta
```

(c) Recorre utilizando bucles *for* las componentes principales (también llamadas *loadings*) del modelo PCA e imprime en pantalla sus valores para sacar conclusiones sobre qué variables están incluidas en qué componente principal. ¿Qué variables tienen más influencia en la primera componente? ¿Y en la segunda? **(1 punto)** NM

```
[ ]: # Respuesta
```

1.1.7 Ejercicio 6

Finalmente, en este ejercicio vamos a aplicar una regresión lineal para intentar predecir el 'gross income' de una película utilizando todas estas variables. **(1.75 puntos)**

(a) Utiliza la función de `Scikit-Learn` proporcionada arriba para realizar una regresión lineal para predecir el 'gross income' según el resto de variables numéricas. Divide los datos en test y train (con un porcentaje de 20% para el test) y luego evalúa los resultados en los datos de test, imprimiendo el pantalla el R2 del modelo. ¿Cómo lo interpretas? **(0.5 puntos)** EG

```
[ ]:  # Respuesta
```

(b) Queremos ver ahora si podemos utilizar el género de la película para predecir juto con el resto de "features" el 'gross income' de las películas, ya que pensamos que podría tener una gran influencia. Modifica tu código anterior ligeramente para añadir como variable predictora el género junto al resto de variables numéricas. Luego intenta hacer la regresión lineal. Te devolverá un error. ¿Por qué ocurre esto? **(0.25 puntos)** NM

[ ]:
```
# Respuesta
```

(c) Claramente lo tenemos que hacer de otra manera. Vamos a crear "dummy variables" para los datos del género de la película y poder realizar la regresión lineal. Utiliza la función get_dummies de `pandas` para ello. Investiga en internet para qué sirve este procedimiento de "dummy variables". Una vez hecho esto, vuelve a realizar el modelo de regresión lineal utilizando las variables numéricas y el género de la película e imprime el R2. ¿Merece la pena, viendo este R2, incluir el género en nuestro modelo? **(0.5 puntos)** EG

[ ]:
```
# Respuesta
```

(d) Si ejecutas las celdas anteriores varias veces, verás el que R2 que devuelve puede ser ligeramente diferente cada vez. Esto es porque los datos de train y test se generan aleatoriamente en cada split (a no ser que hayas fijado una random seed) y al hacer ajuste al modelo en datos ligeramente diferentes, los resultados pueden variar un poco. Para ser más robustos, podemos hacer una validación cruzada de `KFold` con la función cross_val_score, `shuffle = True` y 5 validaciones (usa sólo las variables numéricas, sin el género). **(0.5 puntos)** EG

[ ]:
```
# Respuesta
```

1.1.8 Ejercicio opcional

Continuando a partir del ejercicio anterior, queremos hacer un "ablation study" o "estudio de ablación", que consiste en un análisis de la eliminación progresiva de variables o características en un modelo para evaluar su impacto en el rendimiento o la capacidad predictiva del modelo. Utilizando sólo las variables numéricas, haz un estudio de ablación donde realizas la validación cruzada con `KFold` con cinco validaciones de las variables numéricas del dataset, eliminando una diferente en cada iteración. Compara el R2 del modelo reducido (con una variable menos) y el cambio que produce en el R2 con respecto al modelo completo. ¿Qué conclusiones sacas?

[ ]:
```
# Respuesta
```