prog_datasci_7_analisis_entrega

May 3, 2021

1 Programación para *Data Science*

1.1 PEC 7 - Introducción al análisis de datos en Python

En este Notebook 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 un 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 PAC sin necesidad de hacer este ejercicio. El objetivo de este ejercicio es que sirva 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.

1.2 Ejercicio 1

En esta PEC vamos a trabajar con un conjunto de datos (*dataset*) dedicado al **reconocimiento de dígitos manuscritos** y que es directamente accesible desde la librería de `sklearn`. En este ejercicio haremos una primera *exploración* del *dataset* para tener una mejor comprensión de los datos que vamos a manipular. Empecemos cargando el *dataset*:

[6]:
```
from sklearn import datasets
digits = datasets.load_digits(as_frame=True)
```

(a) Imprime por pantalla el tipo y las dimensiones de los distintos atributos de la variable `digits`. ¿Cuántos *features* por muestra caracterizan este *dataset*? ¿Qué contiene *data* y en qué se diferencia del atributo *images*? Utiliza la función `imshow` de `matplotlib` para visualizar las diez primeras imágenes del *dataset*. NM **(0.5 puntos)**

NOTA: Encontrarás más información sobre el *dataset* en este enlace: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html

(b) ¿Qué finalidad tiene el atributo *target*? ¿Para cuántos dígitos (clases) disponemos de información? Imprime por pantalla el número de muestras por clase y razona si el *dataset* está bien balanceado. EI **(0.5 puntos)**

[1]:
```
# Respuesta
```

1.3 Ejercicio 2

Continuemos *explorando* el conjunto de datos.

(a) ¿Qué rango de valores puede tomar un píxel? Agrupa las imágenes por dígito y calcula, para cada grupo, la varianza por píxel. En el siguiente enlace encontraréis un ejemplo de cómo calcular la varianza utilizando la librería `Pandas`: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.var.html EG **(0.25 puntos)**

(b) Traza los valores de varianza de todos los dígitos en un mismo gráfico, es decir, un gráfico cuyo eje de ordenadas mida la varianza por dígito y cuyo eje de abscisas contenga los 64 *features* posibles de cada muestra del conjunto de datos. NM **(0.25 puntos)**

(c) Crea un dataframe que contenga la *intensidad* de cada imagen (que definiremos como la media del valor de los **cuatro píxeles centrales** de cada imagen: *pixel_3_3, pixel_3_4, pixel_4_3, pixel_4_4*) y el dígito al que está asociado. Muestra por pantalla las 15 primeras filas. NM **(0.5 puntos)**

(d) Visualiza en un **boxplot** las *intensidades* por dígito. Del dataframe anterior, elimina los ***outliers*** (es decir, los *fliers points* según la terminología de la documentación) y muestra por pantalla las dimensiones del dataframe resultante. Para los límites de los *whiskers* podéis tomar los valores por defecto: `whis=1.5`. EI **(1.0 puntos)**

(e) Traza el **histograma** de intensidades por dígito del dataframe **sin *outliers***. A primera vista, para este conjunto de datos nuevo, ¿te parece la *intensidad*, tal y como la hemos definido, un atributo/*feature* útil para mejorar el reconocimiento de algún dígito concreto? NM **(0.5 puntos)**

(f) A la vista de los resultados anteriores, define un criterio basado en la *intensidad* y comprueba si, con algún dígito, se pueden obtener tasas de éxito de más del 90%. Se entiende aquí por tasa de éxito, el porcentaje de casos reconocidos correctamente, sobre el número total de muestras evaluadas. NM **(opcional)**

[2]:
```
# Respuesta
```

1.4 Ejercicio 3

Hemos visto anteriormente que no todos los píxeles aportan la misma información para el reconocimiento de dígitos. En este ejercicio realizaremos un **análisis de componentes principales (PCA)** para saber si podemos reducir el número de *features* del conjunto de datos.

(a) Crea un modelo de la clase `PCA` sin especificar el número de componentes a retener. Entrénalo con el *dataset* original y verifica que su número de componentes corresponde al número de *features* por imagen. EG **(0.5 puntos)**

(b) Consulta la documentación para conocer el detalle de los atributos de la clase y responde a las preguntas siguientes. ¿Cuántas componentes son necesarias para explicar como mínimo el 80% de la varianza del conjunto de datos inicial? ¿Cuántas dimensiones podríamos reducir sin perder información? EG **(0.5 puntos)**

[3]:
```
# Respuesta
```

1.5 Ejercicio 4

En este ejercicio, nos basaremos en una técnica de **aprendizaje supervisado** para reconocer imágenes de dígitos manuscritos. Para ello utilizaremos un modelo de **regresión logística** (`LogisticRegression`) disponible desde la librería de `sklearn`: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

(a) ¿El reconocimiento de imágenes de dígitos es un problema de regresión o un problema de clasificación? EI **(0.25 puntos)**

(b) ¿Podemos utilizar un modelo de regresión logística para el reconocimiento de más de dos clases distintas (10 dígitos)? Justifica brevemente tu respuesta. EI **(0.25 puntos)**

(c) Utiliza la función `train_test_split` para seleccionar el conjunto de datos que servirán para la fase de entrenamiento (***training***) y para la fase de evaluación (***test***). Coged, por ejemplo, 20% del total de muestras para el *test*. A continuación, podéis crear el modelo de `LogisticRegression` (se aconseja pasar como argumento el *solver* `'newton-cg'`) y entrenarlo. Mostrad por pantalla el nivel de exactitud (*accuracy*) obtenida cuando se aplica el modelo sobre el conjunto de *test*. Recuerda que el mismo modelo dispone de la función `score` para el cálculo de la exactitud. EG **(0.5 puntos)**

(d) Fíjate que el *score* cambia cada vez que ejecutas el código. Las muestras escogidas en el conjunto de datos de *training* y de *test* no son siempre las mismas. Haz una **validación cruzada** del modelo con la función `cross_val_score` siguiendo un método *k-Fold* (`KFold`) con k=5 y `shuffle=True`. Muestra por pantalla la media de los scores obtenidos y su desviación típica. EG **(1.0 puntos)**

(e) Siguiendo los pasos del ejercicio 3, realiza de nuevo los apartados (c) y (d) anteriores pero, esta vez, entrena y testea el modelo con un conjunto de datos reducido. Utiliza un PCA correspondiente a una varianza explicada del 80% y muestra por pantalla el nuevo score medio. ¿A qué conclusión llegas? EG **(1.0 puntos)**

[4]:
```
# Respuesta
```

1.6 Ejercicio 5

En este ejercicio, nos planteamos si podríamos reconocer todos los dígitos de manera **no supervisada** agrupando los dígitos por ***clusters***.

(a) Comenta el código siguiente y muestra por pantalla el nivel de exactitud (*accuracy*) del modelo ***K-means*** considerado. NM **(1.0 puntos)**

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=10, random_state=0)
digit_clusters = kmeans.fit_predict(digits.data)

consistent_labels = np.zeros(len(digit_clusters))
for l in set(kmeans.labels_):
    if int(l) > -1:
        ind = (clusters == int(l))
        consistent_labels[ind] = digits.target.loc[ind].mode()
    else:
        print('Algunas imágenes no se han podido clusterizar.')
```

(b) ¿Para qué sirve una **matriz de confusión**? Visualiza la matriz de confusión asociada al resultado del apartado anterior. Para ello, puedes utilizar la clase `ConfusionMatrixDisplay` de `sklearn` cuya documentación encontraréis en el siguiente enlance: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html. EG **(0.5 puntos)**

(c) Observando la matriz de confusión, ¿sabrías decir qué dígito presenta un mayor número de errores? Visualiza los centroides asociados a cada cluster y justifícalo de manera cualitativa. NM **(1.0 puntos)**

(d) En los ejercicios anteriores, hemos utilizado el PCA como técnica para reducir las dimensiones de un dataset. En este apartado, vamos a probar una técnica distinta, el t-SNE (*t-distributed stochastic neighbor embedding*), un potente algoritmo desarrollado para la visualización de datos que permite reducir un dataset de gran dimensión a un dataset de 2 a 3 dimensiones. Utiliza la clase `TSNE` de `sklearn` (https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html) para reducir el conjunto de datos inicial y calcula el *score* de *accuracy* sobre el dataset reducido. ¿Hemos conseguido mejorar la *accuracy* del modelo? EG **(opcional)**

[5]:
```
# Respuesta
```