Análisis de datos con Python

Sistemas Inteligentes 2022-2023

El objetivo de esta práctica es resolver un problema de análisis de datos mediante el lenguaje de programación Python, el cual es uno de los más usados hoy en día para este tipo de proyectos. En este ejercicio, se aplicarán los conocimientos vistos en las **prácticas 6, 7 y opcionalmente 8**. Cada grupo de trabajo usará un conjunto de datos diferente, en concreto, el mismo que le fue asignado para el segundo ejercicio de evaluación. El fichero proporcionado se encuentra en formato Weka (.arff) y aunque en Python es posible leer ficheros de Weka, es más conveniente convertirlo a un formato estándar (como .csv), para lo que se proporcionan instrucciones al final de este documento (**Anexo 1**). *Si por alguna circunstancia no puede trabajar con el fichero asignado o presenta alguna dificultad inesperada, póngase en contacto con el profesor.*

En esta práctica se espera que el alumno siga una metodología similar a la del ejercicio anterior, pero esta vez usando las herramientas que proporciona Python:

- Manipulación de conjuntos de datos mediante *Pandas* y *Numpy* cuando sea necesario.
- Preprocesado de datos y entrenamiento y validación de modelos predictivos mediante *Scikit-learn*.
- Matplotlib (**OPCIONAL**) para representación gráfica de datos. Dado que no se ha visto en clase, se menciona en este enunciado por completitud y se deja como algo voluntario (que se valorará positivamente, pero no penalizará si no se usa).

Metodología:

El trabajo a realizar sobre el conjunto de datos disponible, cubrirá los diferentes pasos necesarios para resolver un problema de clasificación, desde la carga de datos hasta la obtención de los modelos predictivos. Recordemos que el objetivo final es obtener un modelo con la mayor capacidad predictiva posible, por lo que debemos probar diferentes opciones. A continuación se enumeran las distintas etapas de un proyecto de datos y las técnicas que pueden ser usadas en las mismas (no es necesario usarlas todas):

- Tratamiento de datos: Algunas operaciones serán necesarias para poder ejecutar los algoritmos de entrenamiento, otras, sin embargo, son opcionales y nos permitirán, en algunos casos, mejorar la capacidad predictiva de nuestro modelo:
  - Tratamiento de valores faltantes (necesario en aquellos conjuntos de datos en los que existan). Recordemos que según el caso, podemos eliminar filas, eliminar columnas o sustituir los valores faltantes (mediante *SimpleImputer*) por la media, mediana o moda.
  - Conversión de variables en formato texto a numéricas (necesario para cualquier algoritmo de Scikit-learn): Esto se puede hacer mediante *OrdinalEncoder (o*

LabelEncoder) o aplicando una codificación *one-hot (binarización) mediante OneHotEncoder*.

- Normalización, estandarización y otras formas de escalado (opcional, aunque recomendable para algunos algoritmos): Nos permite unificar el rango de valores de las variables. Para esto, existente diferentes operadores como: *Normalizer*, *StandardScaler*, *MinMaxScaler* (en la práctica 7 se proporciona un ejemplo de uso), *MaxAbsScaler*, etc.
- Discretización, es decir, conversión de variables continuas a categóricas (opcional). Esto se puede hacer mediante *KbinsDiscretizer*

- Entrenamiento de modelos predictivos. A continuación se enumeran los algoritmos vistos en clase, disponibles en Scikit-learn:
  - Árboles de decisión (*DecisionTreeClassifier*). De forma opcional podrán usarse otros algoritmos basados en árboles como los bosques aleatorios (*RandomForestClassifier*) o el algoritmo de potenciación del gradiente (*GradientBoostingClassifier*).
  - K vecinos más cercanos (KneighborsClassifier).
  - Naive Bayes (*GaussianNB* o *MultinomialNB*). Recordemos que para aplicar Naive Bayes Multinomial correctamente, los datos deben cumplir ciertas propiedades
  - Redes Neuronales (*MLPClassifier*)
  - K-medias (*Kmeans*). Nótese que esta técnica, al contrario que las anteriores, es de aprendizaje NO supervisado y, por tanto, no permite obtener un modelo predictivo. Es decir, proporciona agrupaciones de instancias de datos con características similares en lugar de predicciones.
  - Otros no vistos en clase. Si tiene dudas sobre otros algoritmos, consulte con el profesor.
- Metodología de valuación de modelos. Hemos visto 3 formas diferentes de evaluar modelos predictivos (cada una con su finalidad, ventajas e inconvenientes):
  1. Entrenamiento y evaluación sobre el conjunto de entrenamiento.
  2. División del conjunto de datos en conjuntos de evaluación y prueba (*train_test_split*).
  3. Evaluación mediante validación cruzada (*cross_val_score*).
- Métricas de evaluación.
  - Hemos visto dos métricas de evaluación, la tasa de aciertos (*accuracy*) y la tasa de aciertos balanceada (*balanced_accuracy*).
  - La siguiente tabla muestra otras métricas para clasificación que proporciona Scikit-learn: https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter
  - Aunque no es una métrica en sí, la matriz de confusión (confusion_matrix) nos proporciona información sobre en qué clases acierta o falla más nuestro modelo.
- Selección de hiperparámetros. Como se vio tanto en las prácticas de Scikit-learn como en las de Weka, el rendimiento de los algoritmos es sensible los valores de los hiperparámetros usados, los cuales deben ser ajustados para cada algoritmo y conjunto de

datos. Ejemplos de parámetros que son importantes para los resultados, son el número de vecinos K a considerar en el algoritmo Knn o la profundidad máxima y el número máximo de hojas en el caso de los árboles de decisión. En el caso de las redes neuronales, además de parámetros importantes como la tasa de aprendizaje o el número de iteraciones, debemos definir la topología de la red, es decir, número de capas ocultas y número de neuronas en cada una de estas.

  - Para encontrar los mejores hiperparámetros, debemos realizar experimentos mediante validación cruzada, probando diferentes valores y seleccionando los que proporcionen mejor rendimiento.
  - Podemos agilizar esta tarea usando la clase *GridSearchCV* que nos permite especificar los diferentes valores de hiperparámetros que deseamos probar para que el sistema evalúe las diferentes combinaciones mediante validación cruzada: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
  - En la práctica 7 se proporciona un ejemplo de uso.
  - En el **Anexo 2** de este documento se proporciona información adicional sobre los principales hiperparámetros y algoritmos a considerar.
- Selección de características. Otro aspecto importante a la hora de obtener mejores modelos es la selección del subconjunto de variables más predictivo.
  - Al final de la práctica 7 se proporcionan dos ejemplos de selección de características, siendo el más interesante de estos, la búsqueda secuencial hacia delante o hacia atrás: http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/
  - Para esto usamos el método *SequentialFeatureSelector* de la librería *mlxtend*: http://rasbt.github.io/mlxtend/api_subpackages/mlxtend.feature_selection/#sequentialfeatureselector

Para más información sobre los diferentes operadores y algoritmos:

- Operadores de preprocesado en scikit-learn: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing
- Algoritmos de entrenamiento de modelos predictivos. Nótese que incluyen algoritmos para clasificación y regresión, mientras que nosotros solo trabajaremos sobre problemas de clasificación: https://scikit-learn.org/stable/supervised_learning.html

Entrega:

El trabajo se presentará en un cuaderno de Jupyter en el que se justifique además cada paso realizado, y se muestren y comenten los resultados obtenidos. Debe probar al menos dos algoritmos diferentes (se valorará el uso de redes neuronales, ya sea mediante Scikit-learn o Keras), de los disponibles en Scikit-learn, e intentará mejorarlos, probando diferentes parámetros en la configuración de los mismos, y opcionalmente, seleccionando características, para acabar

seleccionando un mejor modelo final. Recordemos que la forma más *honesta* de comparar modelos predictivos es mediante validación cruzada. El trabajo incluirá además una fase de preprocesado de datos (y de exploración si fuese necesario).

La entrega será un fichero comprimido que contenga un cuaderno de Jupyter junto con todo lo necesario para ejecutar el mismo, sin tener que realizar ningún tipo de ajuste o configuración adicional. Antes de entregar, por favor compruebe que es posible ejecutar todo el código del cuaderno sin que se produzcan errores.

Variable respuesta:

Algunos conjuntos de datos no especifican la variable respuesta (clase o variable a predecir). En ese caso podemos elegir alguna que sea predecible y/o tenga sentido predecir, si tiene dudas consulte al profesor. Adicionalmente, los alumnos cuyo conjunto de datos ya tiene una variable respuesta seleccionada, si lo desean, pueden probar a predecir otras variables. Es recomendable leer la descripción del conjunto de datos para saber cuál es el objetivo. En algunos casos (no siempre) encontraremos información sobre la variable respuesta junto a las palabras clave CLASSTYPE y CLASSINDEX. En otros casos, la variable respuesta se proporciona con el nombre *class*. Un ejemplo de conjunto de datos que no tiene definida la variable a predecir es:
% CLASSTYPE: nominal
% CLASSINDEX: none specific
Nota: La información sobre la variable respuesta debe ser consultada en el fichero .arff, ya que dicha información no estará presente en el fichero .csv exportado.

Para cualquier duda sobre el conjunto de datos recibido o sobre Scikit-learn, póngase en contacto con el profesor. Adicionalmente, se podrá solicitar de forma justificada un cambio de conjunto de datos (por ejemplo, *tengo un conjunto de datos con pocas variables y me gustaría profundizar en la exploración de técnicas de selección de atributos*).

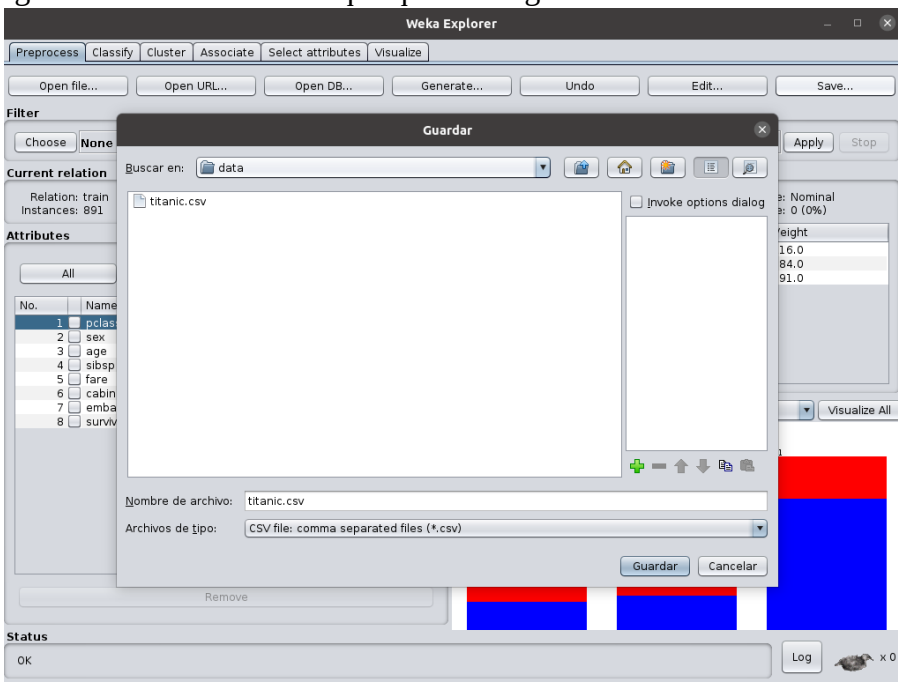
MUY IMPORTANTE:

Cualquier plagio que se detecte en **cualquier parte del trabajo** significará automáticamente la **calificación de cero** en la asignatura para **todos los alumnos involucrados** (la responsabilidad de la copia de una práctica es compartida entre los alumnos). Por tanto, a estos alumnos **no se les conserva**, ni para la actual ni para futuras convocatorias, **ninguna nota** que hubiesen obtenido hasta el momento. Todo ello **sin perjuicio de las** correspondientes **medidas disciplinarias** que se pudieran llevar a cabo.

Anexo 1: Obtención y carga del fichero de datos

1.1 Conversión de fichero de datos desde formato Weka (.arff) a formato CSV estándar.

1. Cargar nuestro conjunto de datos (fichero .arff) en *Weka* usando el botón *Open file*.
2. Hacer clic en el botón *Save* y se abrirá un diálogo como el que vemos a continuación. En el navegamos hasta la ruta en la que queremos guardar el fichero convertido.



3. En la parte inferior seleccionamos el tipo de archivo *CSV file* e introducimos un nombre (con extensión .csv) para el nuevo fichero.
4. Finalmente, hacemos clic en *Guardar*.

1.2 Lectura del fichero guardado con Pandas desde Jupyter:

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('./data/titanic.csv', na_values='?')

In [3]: df.head()
```

Out[3]:

	pclass	sex	age	sibsp	fare	cabin	embarked	survived
0	3	male	22.0	1	7.2500	NaN	S	0
1	1	female	38.0	1	71.2833	C85	C	1
2	3	female	26.0	0	7.9250	NaN	S	1
3	1	female	35.0	1	53.1000	C123	S	1
4	3	male	35.0	0	8.0500	NaN	S	0

```
In [4]: df.isnull().sum()
Out[4]: pclass        0
        sex           0
        age           0
        sibsp         0
        fare          0
        cabin       687
        embarked      2
        survived      0
        dtype: int64
```

La imagen anterior muestra el código necesario para leer el conjunto de datos. Vemos que además de introducir la ruta del fichero (absoluta o relativa), proporcionamos el parámetro *na_values*. Esto se debe a que, como vimos en prácticas anteriores, Weka codifica los valores nulos o faltantes con el carácter '?'. Sin embargo, esa codificación no sigue los estándares, ya que es exclusiva de Weka y, por tanto, debemos indicarlo manualmente. La última línea la usamos para comprobar que se están capturando los valores nulos correctamente (en caso de que estén presentes en nuestro conjunto de datos). Por último, nótese que no incluimos los nombres de las columnas (como vimos en algún ejemplo), ya que en este caso se proporcionan en la primera línea del fichero .csv.

Anexo 2: Hiperparámetros más relevantes.

Se ha comentado que es interesante probar diferentes valores para los parámetros de entrada de los algoritmos, ya que el uso de los valores adecuados proporcionará mejores resultados. Por ejemplo, vimos que limitando la profundidad máxima de crecimiento de un árbol de decisión controlamos el sobreajuste o sobreaprendizaje. Sin embargo, si limitamos la profundidad en exceso, estaremos limitando el aprendizaje del algoritmo. Ambos casos se traducen en resultados pobres a la hora de clasificar datos nuevos o desconocidos. Por esto es importante encontrar los valores más adecuados.

Las implementaciones de los diferentes algoritmos, que Scikit-learn proporciona, tienen muchos parámetros y no se ha profundizado lo suficiente en la materia como para que el alumno sepa cuáles son los más influyentes. Por tanto, a continuación se enumeran los parámetros más importantes para los algoritmos mencionados en el enunciado (no es necesario ni usar todos los algoritmos mencionados y ni probar todos sus parámetros). Además, se proporcionan rangos de valores típicos:

DecisionTreeClassifier

- Véase la documentación oficial para más detalles sobre cada uno de los parámetros: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- Control de la complejidad o crecimiento de diferentes formas mediante los siguientes parámetros. Normalmente, es suficiente usar solo uno: *max_depth*, *min_samples_split*, *min_samples_leaf*. Valores entre 1 y 10 para *max_depth* y entre 1 y 30 para los demás.
- *Criterion*: Medida de impureza para calcular la ganancia de información. Dos posibles valores, *"gini"* y *"entropy"*.

RandomForestClassifier

- Véase la documentación oficial para más detalles sobre cada uno de los parámetros: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- Los mismos que se han mencionado en *DecisionTreeClassifier*.
- *n_estimators*: RandomForest entrena un cierto número de árboles de decisión sobre muestras del conjunto de datos y los promedia. El parámetro *n_estimators* indica el número de árboles de decisión a construir. Valores entre 10 y 1000.

GradientBoostingClassifier

- Véase la documentación oficial para más detalles sobre cada uno de los parámetros: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingCla

ssifier.html

- Los mismos que se han mencionado en *DecisionTreeClassifier*.
- *n_estimators*: GradientBoosting entrena un cierto número de árboles de decisión sobre muestras del conjunto de datos y los va agregando de forma secuencial. El parámetro *n_estimators* indica el número de árboles de decisión a construir. Valores entre 10 y 1000.
- *learning_rate*: Dado que la agregación de árboles es secuencial, la tasa de aprendizaje regula la aportación de cada nuevo árbol construido a la solución final. Valor decimal menor que 1 (algunos valores típicos serían 0.001, 0.05, 0.001, 0.1, etc.). Cuanto menor sea la tasa de aprendizaje, mayor será el número de estimadores que necesitaremos.

KNeighborsClassifier

- Véase la documentación oficial para más detalles sobre cada uno de los parámetros: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- *n_neighbors*: Es el número K de vecinos a considerar para realizar la clasificación. Valor impar mayor o igual a 1 y menor o igual a 15.
- *metric*: Medida de distancia a usar para encontrar los vecinos más cercanos. Valores "*euclidean*", "*manhattan*" o "*hamming*" (esta última solo la usamos en caso de que todas nuestras columnas sean binarias).

GaussianNB

- Véase la documentación oficial para más detalles sobre cada uno de los parámetros: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
- Nada que destacar.

MultinomialNB

- Véase documentación oficial para más detalles sobre cada uno de los parámetros: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
- *alpha:* Suavizado de Laplace visto en clase. Valor mayor que 1 y un cero implica que no se aplicará suavizado.

KMeans

- Véase documentación oficial para más detalles sobre cada uno de los parámetros:

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

- Recordemos que esta técnica pertenece al aprendizaje no supervisado.
- *n_clusters:* Número de clústeres que queremos encontrar. Depende mucho del objetivo. Si trabajamos con datos asociados a problemas de clasificación, K puede ser el número de valores diferentes que tiene nuestra variable respuesta.

Perceptrón multicapa Scikit-learn - MLPClassifier

- Véase documentación oficial para más detalles sobre cada uno de los parámetros: *https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html*
- *hidden_layer_sizes:* Introducimos una tupla con tantos elementos como capas ocultas vaya a tener nuestra red. Cada elemento es el número de neuronas de la capa correspondiente.
- *activation*: Función de activación de las neuronas. Podemos elegir entre las siguientes:
  - identity: Función identidad, es decir, no se hace nada y se devuelve lo mismo que se recibió.
  - logistic: sigmoide.
  - tanh: tangente hiperbólica.
  - relu
- *solver:* Es el algoritmo de entrenamiento de la red. Podemos elegir entre los algoritmos{‘lbfgs’, ‘sgd’, ‘adam’}
- *learning_rate_init:* Tasa de aprendizaje. Adicionalmente, podemos modificar el comportamiento de la tasa de aprendizaje entre constante y adaptativa mediante el parámetro *learning_rate* (posibles valores {‘constant’, ‘invscaling’, ‘adaptive’}).
- *max_iter:* Número máximo de iteraciones que puede ejecutar el algoritmo de entrenamiento.

Recomendación: Para obtener buenos resultados, la mayoría de los algoritmos requieren cierto tratamiento de los datos. Dependiendo de la naturaleza de nuestro conjunto de datos (tipo de las variables, rango de valores, etc.), este tratamiento podrá tener mayor o menor efecto (mejora del rendimiento). En general, es recomendable:

- Unificar los rangos de valores de todas las variables del conjunto de datos. Es decir, aplicar técnicas de escalado, para que todas las variables se encuentren en la misma escala de valores. Los métodos basados en árboles (DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier) no necesitan este tratamiento (tampoco les perjudica).
- Aplicar codificación *one-hot* a las variables categóricas.