Laboratorio 1: Divide y conquista

El objetivo de este trabajo es practicar las estrategias de diseño de algoritmos divide y conquista y búsqueda avara.

El departamento de RRHH de una empresa se encuentre sobrecargado con el número de entrevistas a nuevos empleados que tiene que realizar. Para reducir el número de entrevistas, RRHH decide poblar una BD de empleados entrevistados. En esta BD se indican sus features y si el entrevistado pasó la entrevista o no.

El objetivo de este laboratorio es construir un árbol de decisión que ayude a RRHH a preseleccionar candidatos.

Junto a este documento encontrará el fichero `decision_tree.py` con el prototipo de las funciones a implementar. No debe de modificar el prototipo de estas funciones, solo rellenarlas con la implementación pedida.

Representación de las muestras

En el fichero `decision_tree.py` encontrará una BD `interviewee` con 16 muestras de candidatos, en donde cada muestra está representada con una tupla con 2 valores:

1. Un diccionario de features donde se indica por cada candidato su edad, nivel de inglés, si tiene experiencia y si tiene título universitario. A algunas muestras les faltan features, pero sabemos que los árboles de decisión toleran los valores perdidos.
2. La etiqueta `True`, `False` que clasifica al candidato como idóneo, o no, respectivamente.

```
interviewee = [
     ({'age':52, 'english':'elementary', 'experience':'yes', 'degree':'yes'}, True),
     ({'english':'elementary', 'experience':'no', 'degree':'yes'}, False),
     ({'age':45, 'english':'advanced', 'experience':'yes', 'degree':'no'}, True),
     ({'age':32, 'english':'intermediate', 'experience':'yes', 'degree':'yes'}, True),
     ({'age':62, 'experience':'no', 'degree':'yes'}, False),
     ({'age':21, 'english':'elementary', 'experience':'no', 'degree':'yes'}, False),
     ({'age':55, 'english':'intermediate', 'experience':'yes', 'degree':'yes'}, True),
     ({'age':19, 'english':'advanced', 'experience':'no', 'degree':'no'}, True),
```

```
    ({'age':43, 'english':'elementary', 'experience':'yes', 'degree':'no'}, False),
    ({'age':32, 'english':'advanced', 'experience':'yes', 'degree':'yes'}, True),
    ({'age':60, 'english':'elementary', 'experience':'yes', 'degree':'yes'}, True),
    ({'age':46, 'english':'intermediate', 'experience':'no', 'degree':'no'}, False),
    ({'age':24, 'english':'elementary', 'experience':'no', 'degree':'no'}, False),
    ({'age':18, 'english':'intermediate', 'experience':'no', 'degree':'no'}, False),
    ({'age':59, 'english':'advanced', 'experience':'yes', 'degree':'yes'}, True),
    ({'age':68, 'english':'advanced', 'experience':'no', 'degree':'yes'}, False)
]
```

Tarea 1: Cálculo de la entropía

Implemente la función `entropy(samples)` que recibe una lista de muestras y calcula la entropía de las etiquetas de clase. Esta función debe de tener en cuenta las siguientes consideraciones:

- Si `samples` no tiene muestras, la entropía es 0
- Por convenio, 0 log 0 = 0

En `interviewee` hay 8 `True` y 8 `False`, con lo que su entropía debe ser 1.0:

```
print(entropy(interviewee))
1.0
```

Tarea 2: Algoritmo de partición

Implemente la función `partition(samples, feature, value)` que particiona las muestras `samples` usando la `feature` indicada por el valor dado en `value`. En concreto:

- Las features categóricas se particionan usando el criterio `feature==value`
- Las features numéricas se particionan usando el criterio `feature<=value`

La función ignora las muestras que omitan esa feature, y retorna una tupla con las muestras que cumplen y que no cumplen el criterio de la forma `(part_t,part_f)`.

Por ejemplo, la siguiente partición devuelve 6 muestras que cumplen el criterio de la edad, 9 muestras que no lo cumplen, e ignora la muestra que omite la edad:

```
t_samples, f_samples = partition(interviewee,'age',35)
```

La siguiente partición separa las muestras en las que `'english'=='intermediate'` de las muestras en las que `'english'` toma otro valor, e ignora la muestra que no tiene esta feature:

```
t_samples, f_samples
    = partition(interviewee,'english','intermediate')
```

Tarea 3: Construcción del árbol de decisión

Necesitamos decidir cómo representar el árbol de decisión. Una representación sencilla es un árbol con dos tipos de nodos:

Un `decision_node` es un nodo interno en el que se toma una decisión, y tiene los siguientes atributos:

- `feature`. La feature por la que partir.
- `value`. El valor por el que partir. En caso de ser una feature numérica se usa el criterio `feature<=value`, sino `feature==value`
- `t_branch`. La rama del nodo a seguir cuando el criterio es `True`
- `f_branch`. La rama del nodo a seguir cuando el criterio es `False`

Un `prediction_node` es un nodo hoja que predice la clase de la instancia, y que tiene los siguientes atributos:

- `t_support`. Número de instancias que soportan la clasificación como `True`
- `f_support`. Número de instancias que soportan la clasificación como `False`
- `majority_class()` método que decide como predicción el valor booleano para el que hay más muestras soportándolo

Además, el fichero `decision_tree.py` incluye las siguientes funciones de utilidad:

- `create_majority_prediction_node(samples)` permite crear un `prediction_node` cuyo soporte son las etiquetas de clase de las muestras
- `print_tree(tree)` imprime el árbol generado

Implemente la función `best_partition(samples, split_features)` la cual busca la feature y valor de la partición con mínima impureza. La función devuelve una tupla con la feature, valor y mínima impureza

Por ejemplo, la siguiente ejecución devuelce como feature `'english'`, como valor `'advanced'` y como impureza 0.0:

```
advanced_english = [
      ({'english':'elementary'}, False),
      ({}, False),
      ({'english':'intermediate'}, False),
      ({'english':'advanced' }, True)
]
feature, value, impurity = best_partition(advanced_english, ['age',
'english', 'experience', 'degree'])
```

Después implemente la función `build_tree(samples, split_features)`, la cual genera el árbol de decisión apoyándose en la función `best_partition()`. Un ejemplo de ejecución de esta función sería:

```
english_tree = build_tree(advanced_english, ['age', 'english',
'experience', 'degree'])
print_tree(english_tree)
english : advanced ?
 T->Decision: True (Support T:1, F:0)
 F->Decision: False (Support T:0, F:2)
```

Tarea 4: Clasificación

Implemente la función `classify()` que recibe un diccionario con las features de una observación y predice la clase de la observación. Esta función puede recibir una observación con features omitidas, en cuyo caso devuelve la clase mayoritaria de la rama cuya feature está omitida.

Un ejemplo de ejecución de esta función con features omitidas sería:

```
observation = {'age':45}
print("English: ",
    classify(observation,english_tree).majority_class())
False
```

Forma de entrega

Debe entregar el fichero `decision_tree.py` con la implementación de las funciones descritas en esta tarea. Además debe de responder a las siguientes preguntas:

1. Represente gráficamente el árbol de decisión que seobtiene al ejecutar el siguiente comando:

```
interview_tree = build_tree(interviewee,
    ['age', 'english', 'experience', 'degree'])
print_tree(interview_tree)
```

2. ¿Cuál es el resultado de ejecutar la siguiente clasificación?

```
observation = {'english':'intermediate', 'experience':'yes',
               'degree':'yes'}
prediction = classify(observation, interview_tree)
print(prediction.majority_class())
```

La respuesta a estas preguntas deberá entregarse en formato editables (.doc, .docx, .odf, .rtf). No se aceptan imágenes escaneadas del ejercicio ni el formato .pdf.

Revisión de la actividad

La revisión de las actividades calificables no se realizar por el foro, debe solicitarse a través del tutor.