lime_aleph_ki_campus

January 11, 2021

1 LIME-Aleph

1.0.1 KI-Campus Aufgabe

Willkommen zum Arbeitsauftrag für das Modul **LIME-Aleph** im KI-Campus. Hier werden Sie den typischen Ablauf zum Finden einer symbolischen Erklärung für Black-Box Netzwerke mithilfe der LIME-Aleph Bibliothek Stück für Stück erarbeiten.

Wir wollen zunächst mal die nötigen Bibliotheken importieren und einige nutzerdefinierbare Parameter erzeugen. Eine zu klassifizierende Bilddatei sowie ein vortrainiertes Modell sind schon vorhanden.

[ ]:
```
from skimage.util import img_as_float32
from skimage.transform import resize
from train_model import own_rel
import os, sys, inspect
current_dir = os.path.dirname(os.path.abspath(inspect.getfile(inspect.
 ↪currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0, parent_dir)
from skimage import io
from skimage.io import imshow, show, imsave
import shutil

import lime_aleph as la


IMAGE_FILE = "./pos9000.png" # The path to the image file to be classified by␣
 ↪the black-box
MODEL = "../models_to_explain/model_tower.h5" # The path to the pre-trained␣
 ↪model
K = 3 # The number of important superpixels to be used for perturbation
N = 1000 # The sample size for LIME
OUTPUT_DIR = "../output/" # A path for a directory to save intermediate and␣
 ↪output data
T = 0.8 # The threshold for the binary classifier for when an example is␣
 ↪classified as 'positive'
NOISE = 10 # The allowed false-positive rate for Aleph in percent.
```

Sollte es noch temporäre Daten aus früheren Durchläufen geben, sollen diese nun gelöscht werden:

[ ]:
```
shutil.rmtree(OUTPUT_DIR, ignore_errors=True)
os.makedirs(OUTPUT_DIR)
```

Nun wollen wir das Bild und das vortrainierte Modell in den Speicher laden:

[ ]:
```
image = img_as_float32(io.imread(IMAGE_FILE))
image = resize(image, (own_rel.IMAGE_SIZE, own_rel.IMAGE_SIZE),␣
↪anti_aliasing=True)

model = own_rel.own_rel()
model.load_weights(MODEL)
```

Der nächste Schritt soll nun sein, die im Bild vorhandenen Elemente automatisch zu annotieren. Benutzen Sie hierfür die Funktion **annotate_image_parts** aus dem bereits importierten **lime_aleph** package mit den benötigten Parametern:

[ ]:
```
#[SOLUTION]

annotated_image = la.annotate_image_parts(image, model, OUTPUT_DIR, N)
```

Nachdem das Bild nun annotiert ist (als Annotation wurden auch die Gewichte von LIME für die einzelnen Elemente gefunden), können wir nun die wichtigsten **K** Bildelemente mit der Funktion **find_important_parts** finden. Anschließend können Sie auch die Relationen zwischen den Bildteilen mit der Funktion **find_spatial_relations** finden lassen:

[ ]:
```
#[SOLUTION]

important_superpixels = la.find_important_parts(annotated_image, K)
relations = la.find_spatial_relations(important_superpixels)
```

Die Liste, welche von der Funktion zum Finden von Relationen zurückgegeben wurde, beinhaltet Objekte vom Typ **Relation**. Hier geben wir nun beispielhaft die Informationen der ersten Relation aus. Natürlich müssen Sie den Namen der Liste an Ihre Implementation anpassen.

[ ]:
```
print("Name:", relations[0].name)
print("Start:", relations[0].start)
print("To:", relations[0].to)
```

Der Name beschreibt das Prädikat der räumlichen Relation. Die weiteren Informationen beschreiben die Indices der Start- und Zielelemente der Relation innerhalb des Bildes.

Nun wollen wir das perturbierte Datenset für LIME-Aleph generieren lassen. Benutzen Sie hierzu die Funktion **perturb_instance** mit den erforderlichen Parametern. Lassen Sie sich auch ausgeben, wie viele Instanzen im neuen Datenset sind (Es wird eine Liste mit Instanzen zurückgegeben).

[ ]:
```
#[SOLUTION]
perturbed_dataset = la.perturb_instance(annotated_image, relations, model, T)
print("Number of perturbed instances:", len(perturbed_dataset))
```

Das ILP-Framework Aleph benötigt mehrere Hilfsdateien, die mit der Funktion **write_aleph_files** erzeugt werden. Rufen Sie diese Funktion auf. Es sollen alle räumlichen Relationen verwendet werden! Zur Verfügung stehen folgende Relationen: *left_of*, *right_of*, *top_of*, *bottom_of*, *on*, *under*

[ ]:
```
#[SOLUTION]
used_relations = None # 'None' if you want to allow all relations, otherwise␣
 ↪list with following possibilities: ["left_of", "right_of", "top_of",␣
 ↪"bottom_of", "on", "under"]
la.write_aleph_files(annotated_image, perturbed_dataset, used_relations,␣
 ↪OUTPUT_DIR, NOISE)
```

Schlussendlich muss nun der Induktionsprozess von Aleph angestoßen werden. Dieser Schritt (mit der Funktion **run_aleph**) gibt auch die gefundene Erklärung aus:

[ ]:
```
#[SOLUTION]
la.run_aleph(OUTPUT_DIR)
```

Die Erklärung in Form von Regeln kann nun im angegebenen Ordner in der Datei *explanation.txt* gefunden und interpretiert werden.