code

May 6, 2024

[3]:
```
# install some packages
#pip install svg.path
#pip install tslearn
#pip install threadpoolctl==3.1.0
```

```
Collecting svg.path
  Obtaining dependency information for svg.path from https://files.pythonhosted.
org/packages/d6/ea/ec6101e1710ac74e88b10312e9b59734885155e47d7dbb1171e4d347a364/
svg.path-6.3-py2.py3-none-any.whl.metadata
  Downloading svg.path-6.3-py2.py3-none-any.whl.metadata (13 kB)
Downloading svg.path-6.3-py2.py3-none-any.whl (16 kB)
Installing collected packages: svg.path
Successfully installed svg.path-6.3
Note: you may need to restart the kernel to use updated packages.
```

[1]:
```
import xml.etree.ElementTree as ET
import svg.path
import numpy as np
import cv2
import matplotlib.pyplot as plt
import itertools
import copy
import csv

from PIL import ImageOps
from PIL import Image
from tslearn.metrics import dtw
```

[2]:
```
###
# read the .svg and .jpg files to get binarized images

def get_path_box(path: svg.path):
    xmin = float('inf')
    ymin = float('inf')
    xmax = float('-inf')
    ymax = float('-inf')
    for segment in path:
        if isinstance(segment, svg.path.Line):
```

```
            x1, y1 = segment.start.real, segment.start.imag
            x2, y2 = segment.end.real, segment.end.imag
            xmin = min(xmin, x1, x2)
            ymin = min(ymin, y1, y2)
            xmax = max(xmax, x1, x2)
            ymax = max(ymax, y1, y2)
        elif isinstance(segment, svg.path.QuadraticBezier):
            x1, y1 = segment.start.real, segment.start.imag
            x2, y2 = segment.control.real, segment.control.imag
            x3, y3 = segment.end.real, segment.end.imag
            xmin = min(xmin, x1, x2, x3)
            ymin = min(ymin, y1, y2, y3)
            xmax = max(xmax, x1, x2, x3)
            ymax = max(ymax, y1, y2, y3)
        elif isinstance(segment, svg.path.CubicBezier):
            x1, y1 = segment.start.real, segment.start.imag
            x2, y2 = segment.control1.real, segment.control1.imag
            x3, y3 = segment.control2.real, segment.control2.imag
            x4, y4 = segment.end.real, segment.end.imag
            xmin = min(xmin, x1, x2, x3, x4)
            ymin = min(ymin, y1, y2, y3, y4)
            xmax = max(xmax, x1, x2, x3, x4)
            ymax = max(ymax, y1, y2, y3, y4)
    return xmin, ymin, xmax, ymax
```

[3]:
```
def read_svg_file(filenumber):
    # image_path = 'ground-truth/locations/' + str(filenumber) + '.svg'
    image_path = "/Users/april/Downloads/KWS/locations/" + str(filenumber) + ".
 ↪svg"
    with open(image_path, 'r') as f:
        svg_data = f.read()
    root = ET.fromstring(svg_data)
    # Extract the path commands for each word
    words = []
    ids = []
    for path in root.findall('.//{http://www.w3.org/2000/svg}path'):
        commands = path.attrib['d']
        id = path.attrib['id']
        words.append(commands)
        ids.append(id)
    return words, ids
```

[4]:
```
def get_images_from_words(filenumber, words):
    img = Image.open("/Users/april/Downloads/KWS/images/" + str(filenumber) + ".
 ↪jpg")
    # Extract the word polygons and images
    word_polygons = []
```

```
    word_images = []
    word_id = []
    for i, word in enumerate(words):
        path = svg.path.parse_path(word)
        # Get the bounding box coordinates for the word
        # I tried to find a built-in function but did not find it, so I␣
 ↪implemented manually
        xmin, ymin, xmax, ymax = get_path_box(path=path)
        # Crop the image for the word
        word_img = img.crop((xmin, ymin, xmax, ymax))
        word_arr = np.array(word_img)
        word_polygons.append(path)
        word_images.append(word_arr)
    return word_polygons, word_images
```

[23]:
```
def get_binarized_images(word_images):
    # Binarization
    binarized_word_images = []
    for word_img in word_images:
        #normalize to a consistant image size
        target_size = (120, 90)
        word_img = cv2.resize(word_img, target_size)
        # Convert the image to grayscale
        word_img_gray = ImageOps.grayscale(Image.fromarray(word_img))
        # Adjust this threshold value as needed
        threshold = 128
        word_img_bw = ImageOps.invert(word_img_gray).point(lambda x: 0 if x <␣
 ↪threshold else 255, '1')
        binarized_word_images.append(np.array(word_img_bw))
    return binarized_word_images
```

[6]:
```
###
# use sliding window to create feature matrices

def fraction_of_black_pixels(window : np.array) -> float:
    # gets the proportion of black vs white pixels
    temp_window = window.flatten()
    return temp_window.sum()/temp_window.shape[0]

def upper_conture_location(window : np.array) -> int:
    #finds the highest black pixel in the window
    pos = window.shape[0] - 1
    while(pos > 0):
        if(window[pos].any()):
            break
        pos -= 1
    return pos
```

```
def lower_conture_location(window : np.array) -> int:
    #finds the lowest black pixel in the window
    pos = 0
    while(pos < window.shape[0]):
        if(window[pos].any()):
            break
        pos += 1
    return pos
```

[7]:
```
def fraction_of_black_pixels_between_contures(window : np.array, lower_conture,␣
↪upper_conture) -> float:
    if lower_conture > upper_conture:# if there was no black pixel
        return 0.0
    elif lower_conture == upper_conture:# if there was only one black pixel
        return 1 / window.shape[0]*window.shape[1]
    else:
        return fraction_of_black_pixels(window[lower_conture : upper_conture])

def num_of_transitions(window : np.array, lower_conture : int, upper_conture :␣
↪int) -> int:
    # traverse down the middle of the window and count the number of times␣
↪there is a change from black to white or white to black
    if (lower_conture > upper_conture): # in case the window is empty
        return 0
    else:
        y_axis = window.shape[1]//2
        pos = 0
        transistions = 0
        last_point = window[pos,y_axis]
        while(pos < window.shape[0]):
            cur_point = window[pos,y_axis]
            if(last_point != cur_point):
                transistions += 1
            last_point = cur_point
            pos += 1
        return transistions
```

[8]:
```
def sliding_window(input_image : np.array, window_length : int, window_off_set :
↪ int) -> np.array:
    # creates a feature matrix using a sliding window
    pos1 = 0
    pos2 = window_length
    #print(input_image.shape)
    out_windows = []
    while(pos2 < input_image.shape[1]):
        temp_window = input_image[:, pos1 : pos2]
```

```
        feature_window = []
        upper_conture = upper_conture_location(temp_window)
        feature_window.append(upper_conture)
        lower_conture = lower_conture_location(temp_window)
        feature_window.append(lower_conture)
        feature_window.append(fraction_of_black_pixels(temp_window))
        feature_window.
 ↪append(fraction_of_black_pixels_between_contures(temp_window, lower_conture,␣
 ↪upper_conture))
        feature_window.append(num_of_transitions(temp_window, lower_conture,␣
 ↪upper_conture))
        pos1 = (pos2 + window_off_set)
        pos2 += (window_length + window_off_set)
        out_windows.append(feature_window)
    return np.array(out_windows)
```

[9]:
```
def feature_matrices(binarized_word_images : np.array) -> np.array:
    feature_matrices = []
    for pic in binarized_word_images:
        feature_matrices.append(sliding_window(pic, 1, 1))
    return feature_matrices
```

[10]:
```
###
# Get Feature Matrices for Train and Validation Sets

def get_feature_matrices_train_set(train_files):
    train_set_feature_matrices = []
    ids = []

    for train_file in train_files:
        words, id = read_svg_file(train_file)
        word_polygons, word_images = get_images_from_words(train_file, words)
        binarized_word_images = get_binarized_images(word_images)
        train_set_feature_matrices.
 ↪append(feature_matrices(binarized_word_images))
        ids.append(id)

    train_set_flat = list(itertools.chain.
 ↪from_iterable(train_set_feature_matrices))
    train_ids_flat = list(itertools.chain.from_iterable(ids))
    return train_set_flat, train_ids_flat
```

[11]:
```
def get_feature_matrices_validation_set(validation_files):
    validation_set_feature_matrices = []
    ids = []

    for validation_file in validation_files:
```

```
        words, id = read_svg_file(validation_file)
        word_polygons, word_images = get_images_from_words(validation_file,␣
 ↪words)
        binarized_word_images = get_binarized_images(word_images)
        validation_set_feature_matrices.
 ↪append(feature_matrices(binarized_word_images))
        ids.append(id)

    validation_set_flat = list(itertools.chain.
 ↪from_iterable(validation_set_feature_matrices))
    validation_ids_flat = list(itertools.chain.from_iterable(ids))
    return validation_set_flat, validation_ids_flat
```

[12]:
```
# This function calculates the DTW distance between each word in the validation␣
 ↪set and the train set.
def find_dtw(validation_set, train_set):
    dtw_matrix = np.zeros(shape = (len(train_set), len(validation_set)))
    for i in range(0 , len(train_set)):
        for j in range(0, len(validation_set)):
            dtw_matrix[i, j] = dtw(train_set[i], validation_set[j],␣
 ↪global_constraint="sakoe_chiba")
    return dtw_matrix
```

[13]:
```
def print_word(id):
    image_number = id.split('-')[0]
    words, ids = read_svg_file(image_number)
    idx = None
    for index, string in enumerate(ids):
        if string == id:
            idx = index
            break
    word = words[idx]
    path = svg.path.parse_path(word)
    xmin, ymin, xmax, ymax = get_path_box(path=path)
    img = Image.open('images/' + str(image_number) + '.jpg')
    word_img = img.crop((xmin, ymin, xmax, ymax))
    return(word_img.show())
```

[14]:
```
def rank_dtw_distances(dtw_distances):
    ranked_dtw_distances = np.argsort(dtw_distances, axis = 1)
    return ranked_dtw_distances
```

[27]:
```
def read_transcription(train_files, validation_files):
    f = open("/Users/april/Downloads/KWS/task/transcription.tsv", 'r')
    Lines = f.readlines()
    train_transcription = []
    validation_transcription = []
```

```
    for line in Lines:
        file_number = int(line[0 : 3])

        line = line.strip()
        line = line[10:]

        if file_number in train_files:
            train_transcription.append(line)
        else:
            validation_transcription.append(line)

    return train_transcription, validation_transcription
```

[16]:
```
def transfrom_rank_into_word(ranked_dtw_distances, train_transcription):
    train_word_ranks = []

    for validation_word_index in ranked_dtw_distances:
        rank_per_word = []
        for ranked_train_word_index in validation_word_index:
            rank_per_word.append(train_transcription[ranked_train_word_index])

        train_word_ranks.append(rank_per_word)

    return train_word_ranks
```

[30]:
```
def read_keywords():
    f = open("/Users/april/Downloads/KWS/task/keywords.tsv", 'r')
    Lines = f.readlines()

    for i, line in enumerate(Lines):
        line = line.strip()

        Lines[i] = line

    return Lines
```

[18]:
```
def calculate_precision_and_recall(precision_top_ranks, keywords,␣
 ↪ranked_train_words, validation_transcription):
    precisions = [1]
    recall = []

    for precision in precision_top_ranks:
        true_positive = 0
        false_positive = 0
        false_negative = 0
        keywords_length = len(keywords)
```

```
        for keyword in keywords:
            if keyword in validation_transcription: # relevant elements, the␣
 ↪ones we actually find in both the train and validation set
                index = validation_transcription.index(keyword)
                top_precision_words = ranked_train_words[index][:precision]
                if (keyword in top_precision_words):
                    true_positive += 1
                    false_positive += precision - 1
                else:
                    false_positive += precision
                    false_negative += 1
            else:
                keywords_length -= 1

        if precision == precision_top_ranks[0]: # only print this once
            print("Keywords actaully found in validation set: " +␣
 ↪str(keywords_length))
            print("Number of total Keywords: " + str(len(keywords)))

        precisions.append(true_positive/(true_positive+false_positive))
        recall.append(true_positive/(true_positive+false_negative))

    recall.append(1)
    return precisions, recall
```

[19]:
```
def draw_precision_recall_curve(precision, recall):
    plt.plot(recall, precision)
    plt.title("Precision-Recall Curve")
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.show()
```

[20]:
```
validation_word_similarity = {
    "dtw": 0,
    "word_id": ""
}

training_word = {
    "transcription": "",
    "similarities": []
}


# This function calculates the DTW distance between each word in the validation␣
 ↪set and the train set.
def find_dtw_competition(validation_set, train_set, validation_ids,␣
 ↪train_transcription):
```

```
    # dtw_matrix = np.zeros(shape = (len(train_set), len(validation_set)))
    dtw_matrix = []
    for i in range(0 , len(train_set)):
        train_word = copy.deepcopy(training_word)
        train_word["transcription"] = train_transcription[i]
        temp_list = []
        for j in range(0, len(validation_set)):
            # dtw_matrix[i, j] = dtw(train_set[i], validation_set[j],␣
↪global_constraint="sakoe_chiba")
            validation_word = copy.deepcopy(validation_word_similarity)
            validation_word["dtw"] = dtw(train_set[i], validation_set[j],␣
↪global_constraint="sakoe_chiba")
            validation_word["word_id"] = validation_ids[j]
            temp_list.append(validation_word)
        sorted_temp_list = sorted(temp_list, key=lambda x: x["dtw"],␣
↪reverse=True)
        train_word["similarities"] = copy.deepcopy(sorted_temp_list)
        dtw_matrix.append(train_word)
    return dtw_matrix
```

[21]:
```
def store_list_csv(input_list):
    file_path = 'my_list_short.csv'

    # Open the CSV file in write mode
    with open(file_path, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerows(input_list)
```

[24]:
```
# read the train and validation words and build feature matrices
train_files = []
validation_files = []
train_files.extend(range(270, 280))
validation_files.extend(range(300, 305))
train_set, train_ids = get_feature_matrices_train_set(train_files)
validation_set, validation_ids =␣
↪get_feature_matrices_validation_set(validation_files)
```

[25]:
```
# Calculate the DTW distances between each word in the validation set and the␣
↪train set and rank them
dtw_distances = find_dtw(validation_set, train_set)
ranked_dtw_distances = rank_dtw_distances(dtw_distances)
```

[28]:
```
train_transcription, validation_transcription = read_transcription(train_files,␣
↪validation_files)
```

[31]:
```
# Compare the ranked words for the keywords and calculate the precision
```

```
train_transcription, validation_transcription = read_transcription(train_files,␣
 ↪validation_files)
ranked_train_words = transfrom_rank_into_word(ranked_dtw_distances,␣
 ↪train_transcription)
keywords = read_keywords()
precision_top_ranks = np.arange(1, len(train_transcription))
precision, recall = calculate_precision_and_recall(precision_top_ranks,␣
 ↪keywords, ranked_train_words, validation_transcription)
```

```
Keywords actaully found in validation set: 35
Number of total Keywords: 35
```

[335]:
```
store_list_csv(main_list)
```

[32]:
```
draw_precision_recall_curve(precision, recall)
```