variety_and_spikiness

February 27, 2025

[1]:
```
'''
"title": "variety_and_spikiness"
"author": "sunnyxxy"
'''

def rao_quadratic_entropy_log(values, log_iterations=1):
    values = np.array(values)

    # Determine the unique categories and their counts
    unique, counts = np.unique(values, return_counts=True)
    p = counts / counts.sum()  # relative frequencies
    distance_func = lambda x, y, log_it: functools.reduce(lambda acc, _: np.
↪log1p(acc), range(log_it), abs(x-y))

    # Compute the distance (dissimilarity) matrix for the unique values
    n = len(unique)
    dist_matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            dist_matrix[i, j] = distance_func(unique[i], unique[j],␣
↪log_iterations)

    # Compute Rao's Quadratic Entropy: Q = sum_{i,j} p_i * p_j * d(i, j)
    Q = np.sum(np.outer(p, p) * dist_matrix)
    return Q

def variety(note_seq, note_seq_by_column): # assume that note_seq already is␣
↪sorted by head
    heads = [n[1] for n in note_seq]
    tails = [n[2] for n in note_seq] # -1 for rice is included
    tails.sort()
    head_gaps = [int(heads[i+1] - heads[i]) for i in range(len(heads)-1)]
    tail_gaps = [int(tails[i+1] - tails[i]) for i in range(len(tails)-1)]
    head_variety = rao_quadratic_entropy_log(head_gaps, log_iterations=1)
    tail_variety = rao_quadratic_entropy_log(tail_gaps, log_iterations=1)

    '''
```

```
    The incorporation of col_variety component is suggested by Anson_98.
    '''
    head_gaps = []
    for k in range(len(note_seq_by_column)):
        heads = [n[1] for n in note_seq_by_column[k]]
        head_gaps = head_gaps + [int(heads[i+1] - heads[i]) for i in␣
 ↪range(len(heads)-1)]
    col_variety = 2.5*rao_quadratic_entropy_log(head_gaps, log_iterations=2)

    return 0.5*head_variety + 0.11*tail_variety + 0.45*col_variety

def spikiness(D_sorted, w_sorted): # w_sorted is also sorted in D
    weighted_mean = (np.sum(D_sorted**5 * w_sorted) / np.sum(w_sorted))**(1 / 5)
    weighted_variance = (np.sum((D_sorted**8 - weighted_mean**8)**2 * w_sorted)␣
 ↪/ np.sum(w_sorted))**(1 / 8)

    return np.sqrt(weighted_variance) / weighted_mean
```

[ ]:

```

```