Raport - Algorytmy Tekstowe - Laboratorium 5

Łukasz Sochacki

Podpunkt 1.

W celu realizacji podpunktu 1 zostały zaimplementowane następujące metryki :

- LCS
- DICE
- Euklidesowa

Metryka LCS została zrealizowana za pomocą poniższej funkcji.

```
def LCSMetric(first, second):
    n = len(first)
    m = len(second)
    table = []
    res = 0
    for i in range(n + 1):
        table.append([])
        for j in range(m + 1):
            table[i].append(0)

    for i in range(n):
        for j in range(m):
            if first[i] == second[j]:
                table[i + 1][j + 1] = table[i][j] + 1
                res = max(res, table[i + 1][j + 1])

    return 1 - res / max(n, m)
```

Następnie została zrealizowana metryka DICE. W tym celu została zaimplementowana funkcja dzieląca dany tekst na ngramy.

```
def ngram(word, n):
    m = len(word)
    s = set()
    for i in range(m - n + 1):
        s.add(word[i:i + n])
    return s
```

Wykorzystując powyższą funkcję następnie została zaimplementowana funkcja wyznaczająca wartość metryki DICE między dwoma słowami.

```
def DICE(first, second):
    x = ngram(first, 2)
    y = ngram(second, 2)
    denominator = len(x) + len(y)
    counter = 0
    for el in x:
        if el in y:
            counter += 1

    return 2 * counter / denominator
```

Metryka Euklidesowa została zaimplementowana za pomocą dwóch funkcji. Pierwsza z nich uzyskuje informacje o danym słowie w postaci krotki zawierającej informacje o ilości wystąpień danych znaków, jakie znaki wystąpiły w danym tekście oraz pierwiastek z sumy wszystkich wystąpień znaków w tekście.

```
def vecFromWord(word):
    unique = Counter(word)
    letters = set(unique)
    occurances = unique.values()
    res = 0
    for val in occurances:
        res += val**2

    return unique, letters, sqrt(res)
```

Wykorzystując powyższą funkcje została zaimplementowana funkcja wyliczająca metrykę Euklidesową.

```
def cosDistance(first, second):
    x = vecFromWord(first)
    y = vecFromWord(second)
    sx = x[1]
    sy = y[1]
    occcurancesx = x[0]
    occcurancesy = y[0]
    both = sx.intersection(sy)
    res = 0
    for el in both:
        res += occcurancesx[el] * occcurancesy[el]

    res /= x[2]
    res /= y[2]
    return res
```

Podpunkt 2.

Poniżej znajduje się funkcja obliczająca jakość klasteryzacji za pomocą indeksu Daviesa-Bouldina.

```
def DaviesBouldin(X, Y):
    n = len(np.bincount(Y))
    clusters = [X[Y == i] for i in range(n)]
    centroids = []
    for el in clusters:
        centroids.append(np.mean(el, axis=0))

    vars = [np.mean([euclidean(el, centroids[i]) for el in tab]) for i, tab
in enumerate(clusters)]
    res = []
    for i in range(n):
        for j in range(n):
            if i == j:
                continue
            else:
                res.append((vars[i] + vars[j]) / euclidean(centroids[i],
centroids[j]))

    val = np.max(res)/n
    return val
```

Następnie zostały zaimplementowane dwie funkcje w celu realizacji indeksu Dunna. Pierwsza z nich służy do wyliczenia najdłuższej odległości między otrzymanymi klastrami.

```
def maxDistance(points):
    if points.shape[0] <= 1:
        return 0
    elif points.shape[0] == 2:
        return ((points[0] - points[1])*(points[0] - points[1])).sum()
    else:
        samples = points[spatial.ConvexHull(points).vertices]
        return spatial.distance_matrix(samples, samples).max()
```

Następnie za pomocą drugiej funkcji uzyskujemy wartość indeksu Dunna.

```
def Dunn(points, labels, centroidslength):
    unique = np.unique(labels)
    maxdistance = max(maxDistance(points[labels == el]) for el in unique)
    mindistance = centroidslength.min()
    return mindistance / maxdistance
```

Podpunkt 3.

W celu realizacji podpunktu została wykorzystana biblioteka **nltk** w celu znalezienia słów w tekście, które nie powinny należeć do stoplisty. Poniższe funkcja zwracają stoplistę.

```
def createStopList(text):
    text = nltk.word_tokenize(text)
    stopw = set(stopwords.words('english'))
    stoplist = []
    for w in text:
        if w in stopw:
            stoplist.append(w)

    return stoplist
```

Podpunkt 4

Podczas realizacji podpunktu udało się jedynie zaimplementować algorytm klasteryzacji otrzymanego tekstu. Poniżej znajduje się funkcja realizująca klasteryzację tekstu za pomocą otrzymanej metryki oraz podanej stoplisty lub jej braku.

```
def clusterization(text, metric, stoplist=None):
    text = text.copy()
    if stoplist is None:
        pass
    else:
        for i, s in enumerate(text):
            words = s.split()
            tab = []
            for el in words:
                if el not in stoplist:
                    tab.append(el)

            text[i] = " ".join(tab)

    clusteredtext = np.array(text, dtype=object).reshape(-1, 1)
    distancetable = cdist(clusteredtext, clusteredtext, metric=lambda
first, second: metric(first[0], second[0]))
    scan = DBSCAN(eps=1, min_samples=2)
    res = scan.fit(distancetable).labels, distancetable
    return res
```